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: Управляющие конструкции языка Python

Студент гр. 3341 _____ Лодыгин И.А.

Преподаватель _____ Иванов Д.В.

Санкт-Петербург

2023

Цель работы

Целью работы является освоение работы с управляющими конструкциями языка Python на примере использующей их программы.

Задание

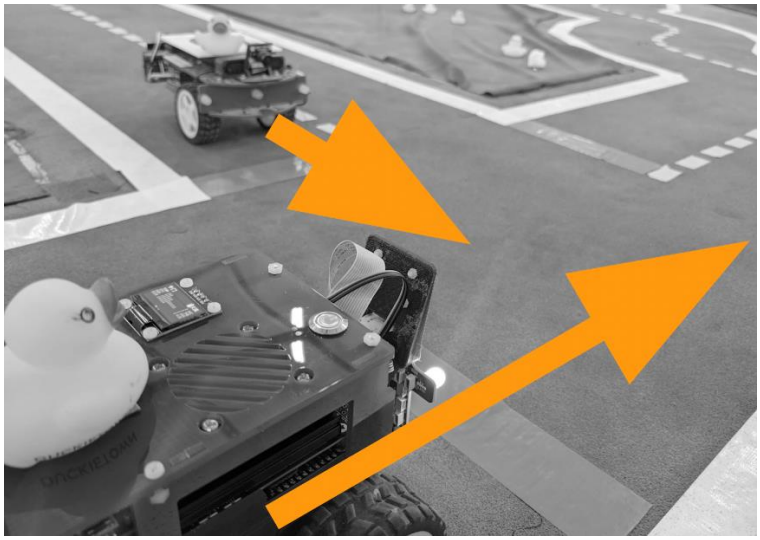
1 вариант.

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. Содержательная постановка задачи

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.

Пример ситуации:



Формальная постановка задачи

Оформите решение в виде отдельной функции check_collision. На вход функции подаются два ndarray -- коэффициенты bot1, bot2 уравнений прямых bot1 = (a1, b1, c1), bot2 = (a2, b2, c2) (уравнение прямой имеет вид ax+by+c=0).

Функция должна возвращать точку пересечения траекторий (кортеж из 2 значений), предварительно округлив координаты до 2 знаков после запятой с помощью round(value, 2).

Пример входных данных:

array([-3, -6, 9]), array([8, -7, 0])

Пример возвращаемого результата:

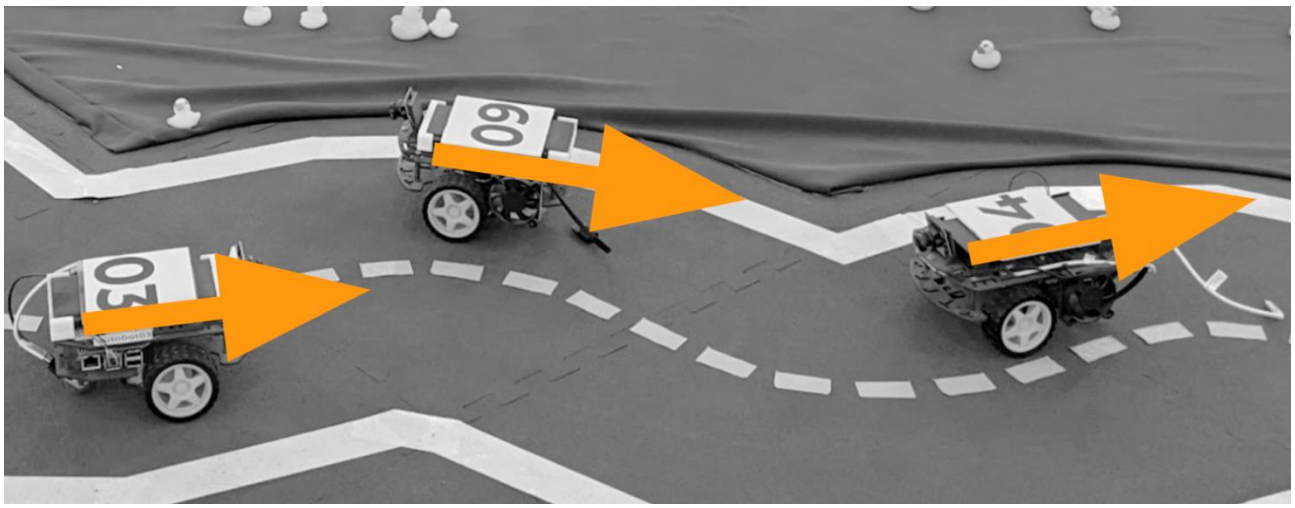
(0.91, 1.04)

Примечание: помните про ранг матрицы и как от него зависит наличие решения системы уравнений. В случае, если решение найти невозможно (например, из-за линейно зависимых векторов), функция должна вернуть None.

Задача 2. Содержательная часть задачи

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 местности, по которой они двигались. Для построения карты местности необходимо знать уравнение плоскости. Ваша задача -- помочь дакиботам найти уравнение плоскости, в которой они двигались.

Пример ситуации:



Формальная постановка задачи

Оформите задачу как отдельную функцию check_surface, на вход которой передаются координаты 3 точек (3 ndarray 1 на 3): point1, point2, point3. Функция

должна возвращать коэффициенты a, b, c в виде ndarray для уравнения плоскости вида ax+by+c=z.

Перед возвращением результата выполнение округление каждого коэффициента до 2 знаков после запятой с помощью round(value, 2).

Например, даны точки: A(1, -6, 1); B(0, -3, 2); C(-3, 0, -1). Подставим их в уравнение плоскости:

$$a \cdot 1 + b(-6) + c = 1$$
$$a \cdot 0 + b(-3) + c = 2$$
$$a(-3) + b \cdot 0 + c = -1$$

Составим матрицу коэффициентов:

$$\begin{pmatrix} 1 & -6 & 1 \\ 0 & -3 & 1 \\ -3 & 0 & 1 \end{pmatrix}$$

Вектор свободных членов:

$$\begin{pmatrix} 1 \\ 2 \\ -1 \end{pmatrix}$$

Для такой системы уравнение плоскости имеет вид: z = 2x + 1y + 5

Пример входных данных:

array([1, -6, 1]), array([0, -3, 2]), array([-3, 0, -1])

Возвращаемый результат:

[2. 1. 5.]

Примечание: помните про ранг матрицы и как от него зависит существование решения системы уравнений. В случае, если решение найти невозможно (невозможно найти коэффициенты плоскости из-за, например, линейно зависимых векторов), функция должна вернуть None.

Задача 3. Содержательная часть задачи

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.

Оформите решение в виде отдельной функции check_rotation. На вход функции подаются ndarray 3-х координат дакибота и угол поворота. Функция возвращает повернутые ndarray координаты, каждая из которых округлена до 2 знаков после запятой с помощью round(value, 2)..

Пример входных аргументов:

array([1, -2, 3]), 1.57

Пример возвращаемого результата:

[2. 1. 3.]

Основные теоретические положения

С помощью строки import numpy as np была подключена библиотека numpy. NumPy — это библиотека языка Python, добавляющая поддержку больших многомерных массивов и матриц, вместе с большой библиотекой высокоуровневых (и очень быстрых) математических функций для операций с этими массивами. В работе были использованы следующие функции данной библиотеки: np.vstack() – вертикальное объединение матриц, np.linalg.matrix_rank() – нахождение решения системы уравнений, np.linalg.matrix_rank – нахождение ранга матрицы, np.round() – округление всех чисел из numpy массива до заданного числа знаков после запятой, np.dot() – перемножение матриц.

Выполнение работы

Происходит импорт библиотеки numpy: import numpy as np.

Для решения первой задачи используется функция check_collision. На вход функции подаются два ndarray - коэффициенты bot1, bot2 уравнений прямых bot1 = (a1, b1, c1), bot2 = (a2, b2, c2) (уравнение прямой имеет вид ax+by+c=0).

В начале была создана матрица matrix, состоящая из первых двух элементов каждого из входных массивов bot1 и bot2 (коэффициентов при x и y). Это было сделано с помощью функции np.vstack(), которая вертикально объединяет матрицы.

Затем был проверен ранг этой матрицы с помощью функции np.linalg.matrix_rank(). Ранг матрицы показывает размерность максимального линейно независимого подмножества столбцов матрицы. Если ранг матрицы не равен 2, то это означает, что траектории ботов параллельны и не пересекаются, поэтому функция возвращает None.

Если ранг матрицы равен 2, то создаётся вектор answer, состоящий из отрицательных значений третьего элемента каждого из входных массивов bot1 и bot2 (коэффициент ‘c’ переносится через равно с противоположным знаком). Затем была использована функция np.linalg.solve() для решения системы линейных уравнений matrix * coordinates = answer. Результатом решения является массив coordinates, содержащий координаты точки пересечения траекторий.

Наконец, были округлены координаты точки до 2 знаков после запятой с помощью функции np.round().

В итоге, функция check_collision выполняет все необходимые операции для решения задачи и возвращает точку пересечения траекторий ботов.

Для решения второй задачи используется функция check_surface, на вход которой передаются координаты 3 точек (3 ndarray 1 на 3): point1, point2, point3. Функция должна возвращать коэффициенты a, b, c в виде ndarray для уравнения плоскости вида ax+by+c=z.

Сначала были заданы координаты трех точек point1, point2 и point3. Затем были сохранены значения координат z для каждой точки в переменные first, second и third. Коэффициенты перед “c” взяты равным единице по умолчанию.

Для решения задачи необходимо найти уравнение плоскости, проходящей через эти три точки. Уравнение плоскости имеет вид ax + by + c = z. Чтобы найти коэффициенты a, b и c, необходимо составить систему линейных уравнений, подставив в нее координаты точек.

Для этого была создана матрица matrix с помощью функции np.vstack(). Функция np.vstack() вертикально объединяет массивы, поэтому каждая строка матрицы matrix содержит кооэффициенты a, b, c.

Затем был выполнен проверка ранга матрицы с помощью функции np.linalg.matrix_rank(). Ранг матрицы показывает размерность максимального линейно независимого подмножества столбцов матрицы. Если ранг матрицы не равен 3, то это означает, что точки лежат на одной прямой и не образуют плоскость, поэтому функция возвращает None.

Если ранг матрицы равен 3, то был создан вектор answer, содержащий значения координат z для каждой точки. Затем была использована функция np.linalg.solve() для решения системы линейных уравнений matrix * plane = answer. Результатом решения является вектор plane, содержащий коэффициенты a, b и c.

Наконец, были округлены значения коэффициентов до 2 знаков после запятой с помощью функции round().

В итоге, функция check_surface выполняет все необходимые операции для решения задачи и возвращает коэффициенты уравнения плоскости.

Для решения третьей задачи используется функция check_rotation. На вход функции подаются ndarray 3-х координат дакибота и угол поворота.

Для выполнения поворота дакибота вокруг оси z на угол rad, была создана матрица turn. Матрица turn представляет собой матрицу поворота в трехмерном пространстве. Каждая строка матрицы turn соответствует одной из осей координат. В первой строке матрицы turn записаны значения cos(rad), sin(rad), 0,

во второй строке - значения -sin(rad), cos(rad), 0, а в третьей строке - значения 0, 0, 1.

Далее была выполнена операция умножения массива vec на матрицу turn с помощью функции np.dot(). Функция np.dot() выполняет умножение двух матриц. В результате получается новый массив coordinates, содержащий повернутые координаты дакибота.

Затем каждая координата в массиве coordinates была округлена до 2 знаков после запятой с помощью функции np.round(). Функция np.round() округляет каждое значение до указанного количества знаков после запятой.

Наконец, полученные координаты были возвращены в качестве результата выполнения функции check_rotation.

Тестирование.

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные	Комментарии
1.	[-3, -6, 9], [8, -7, 0]	(0.91, 1.04)	Задача 1
2.	[-5, 4, 9], [3, -1, 3]	(-3.0, -6.0)	Задача 1
3.	[1, -6, 1], [0, -3, 2], [-3, 0, -1]	[2. 1. 5.]	Задача 2
4.	[1, -2, 3], 1.57	1.57	Задача 3
5.	[5, 4, 2], 1.3	[-2.52 5.89 2.]	Задача 3

Выводы

Была освоена работа с управляющими конструкциями на языке Python на примере использующей их программы. Произошло ознакомление с существующими управляющими конструкциями, с их применением на практике.

Разработаны функции, которые, используя библиотеку numpy и матрицы, позволяют дакиботам определять точку потенциального столкновения, определять карту местности, по которой они двигались, помогают им повернуться в нужное направление.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: LR1CS.py

```
import numpy as np
def check_collision(bot1, bot2):
    matrix = np.vstack((bot1[:2], bot2[:2]))
    if (np.linalg.matrix_rank(matrix) != 2):
        return None
    else:
        answer = [-bot1[2], -bot2[2]]
        coordinates = np.linalg.solve(matrix, answer)
        coordinates = np.round(coordinates, 2)
        return tuple(coordinates)
def check_surface(point1, point2, point3):
    first = point1[2]
    second = point2[2]
    third = point3[2]
    point1[2] = 1
    point2[2] = 1
    point3[2] = 1
    matrix = np.vstack((point1, point2, point3))
    if (np.linalg.matrix_rank(matrix) != 3):
        return None
    else:
        answer = [first, second, third]
        plane = np.linalg.solve(matrix, answer)
        plane[0] = round(plane[0], 2)
        plane[1] = round(plane[1], 2)
        plane[2] = round(plane[2], 2)
        return plane
def check_rotation(vec, rad):
    turn = [[np.cos(rad), np.sin(rad), 0], [-np.sin(rad), np.cos(rad),
0], [0, 0, 1]]
    coordinates = np.dot(vec, turn)
    coordinates[0] = np.round(coordinates[0], 2)
    coordinates[1] = np.round(coordinates[1], 2)
    coordinates[2] = np.round(coordinates[2], 2)
    return coordinates
```