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. 3343 _____ Поддубный В.А.

Преподаватель _____ Иванов Д.В.

Санкт-Петербург

2023

Цель работы

Целью работы являлось изучение и практическое применения принципов программирования на языке Python, при этом используя модуль *numpy*, в частности пакет *numpy.linalg*.

Задание

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.

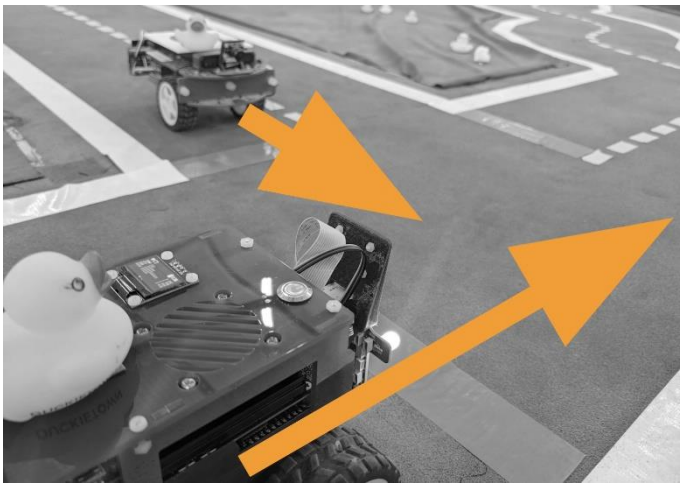

Рисунок 1 – Задача 1

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).

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.

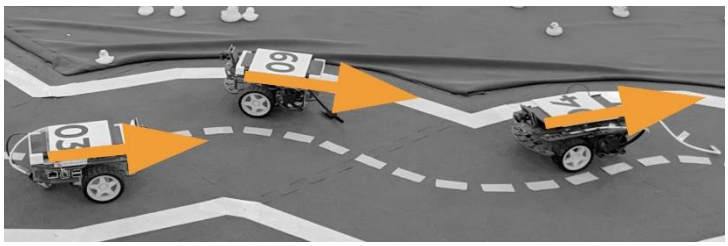

Рисунок 2 – Задача 2

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).

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.

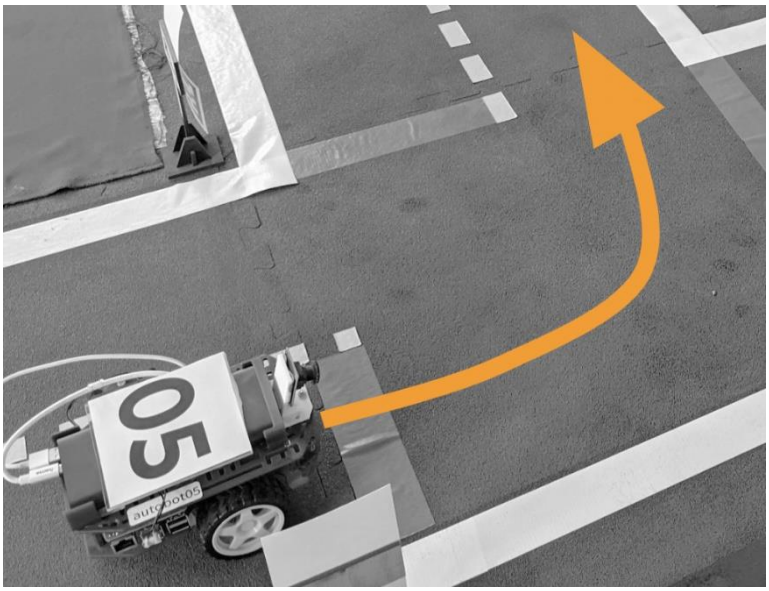

Рисунок 3 – Задача 3

Формальная постановка задачи

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*)..

Выполнение работы

Мой программный код написан на языке Python и использует библиотеку NumPy для решения различных задач, связанных с работой и управлением роботами. Программа содержит три функции, каждая из которых выполняет свою задачу.

1. Функция check_collision(bot1, bot2):

Функция check_collision принимает на вход два аргумента bot1 и bot2, представляющих собой коэффициенты уравнений прямых, определенных ботами.

Внутри функции происходит следующее:

- Коэффициенты A и B для уравнения прямой bot1 и bot2 извлекаются из аргументов bot1 и bot2.

- Коэффициент C для уравнения прямой вычисляется как отрицательное значение третьего коэффициента bot1 и bot2.

- Проверяется, имеют ли две прямые общую точку, используя np.linalg.matrix_rank. Если ранг матрицы коэффициентов не равен 2, то функция возвращает None, что означает, что прямые не пересекаются.

- Если ранг матрицы равен 2, то с помощью np.linalg.solve вычисляется точка пересечения прямых и возвращается в виде кортежа.

2. Функция check_surface(point1, point2, point3):

Функция check_surface принимает на вход три аргумента point1, point2 и point3, представляющих собой трехмерные координаты точек в пространстве.

Внутри функции происходит следующее:

- Извлекаются координаты точек в виде векторов с добавленной третьей координатой, равной 1.

- Проверяется, образуют ли три точки плоскость, используя np.linalg.matrix_rank. Если ранг матрицы точек не равен 3, то функция возвращает None, что означает, что точки не лежат в одной плоскости.

- Если ранг матрицы равен 3, то с помощью np.linalg.solve вычисляются координаты вектора, описывающего плоскость, и возвращаются в виде кортежа.

3. Функция check_rotation(vec, rad):

Функция check_rotation принимает на вход два аргумента: vec - вектор в трехмерном пространстве, и rad - угол в радианах.

Внутри функции происходит следующее:

- Из вектора vec извлекаются только первые две координаты и сохраняются в vect2D.

- Вычисляются две матрицы col1 и col2, представляющие собой матрицу поворота на заданный угол rad.

- С помощью np.linalg.solve вычисляется новый двухмерный вектор rotation, который получается после поворота vec на угол rad.

- Полученный двухмерный вектор rotation дополняется третьей координатой из оригинального вектора vec, и возвращается в виде трехмерного вектора vect3D.

Данный код демонстрирует использование библиотеки NumPy для решения линейных систем уравнений и работы с векторами и матрицами в трехмерном пространстве. Он может быть полезен при решении задач, связанных с управлением роботами и анализом их движения.

Разработанный программный код см. в приложении А.

Тестирование

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные	Комментарии
1.	bot1 = [2 3 -5] bot2 = [-1 4 2]	(-1.0, 2.0)	Тестирование функции *check_collision*
2.	point1 = [1 2 3] point2 = [3 6 9] point3 = [2 4 6]	[1.0, 2.0, 3.0]	Тестирование функции *check_surface*
3.	vec = [3, 3, 3] rad = math.pi / 6	[4.0, 2.0, 3.0]	Тестирование функции *check_rotation*

Выводы

Программа разработана на языке Python с использованием библиотеки NumPy для выполнения операций над геометрическими данными и проведения вычислений. В процессе создания программы были изучены и применены различные аспекты программирования:

- Управляющие конструкции: Программа использует конструкции условия (`if`) для проверки различных условий и принятия решений на основе этих условий. Это обеспечивает гибкость и возможность реагировать на разнообразные ситуации.

- Функции: Для повышения структурированности кода и улучшения его читаемости были созданы и использованы функции. Функциональный подход делает программу более модульной и облегчает понимание кода.

- Библиотека NumPy: Для решения задач, связанных с геометрией и математическими вычислениями, программа активно использовала функциональность библиотеки NumPy. Это включает в себя операции над матрицами, вычисление норм векторов и многие другие вычисления.

- Ввод и вывод данных: Программа взаимодействует с пользователем, ожидая ввода данных, и выводит результаты обработки на экран. Это обеспечивает удобство использования программы.

Таким образом, разработанная программа предоставляет возможность выполнения различных операций над геометрическими данными и проведения численных вычислений. Это важный инструмент для решения задач, связанных с геометрией и математикой, и способствует улучшению понимания и решения подобных задач.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.py

```
import math
import numpy as np


def check_collision(bot1, bot2):
bot1AB = [bot1[0], bot1[1]]
bot2AB = [bot2[0], bot2[1]]
bot1C = -bot1[2]
bot2C = -bot2[2]

if np.linalg.matrix_rank([bot1AB, bot2AB]) != 2:
return None

collision = np.round(np.linalg.solve([bot1AB, bot2AB], [bot1C,
bot2C]), 2)

return tuple(collision)


def check_surface(point1, point2, point3):
vect = [point1[2], point2[2], point3[2]]
point1 = [point1[0], point1[1], 1]
point2 = [point2[0], point2[1], 1]
point3 = [point3[0], point3[1], 1]

if np.linalg.matrix_rank([point1, point2, point3]) != 3:
return None

surface = np.round(np.linalg.solve([point1, point2, point3], vect),
2)

return surface


def check_rotation(vec, rad):
vect2D = [vec[0], vec[1]]
col1 = [math.cos(rad), math.sin(rad)]
col2 = [-math.sin(rad), math.cos(rad)]

rotation = np.round(np.linalg.solve([col1, col2], vect2D), 2)

vect3D = np.array([rotation[0], rotation[1], vec[2]])

return vect3D
```