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. 3341 Шаповаленко Е.В.

Преподаватель Иванов Д.В.

Санкт-Петербург

Цель работы

Цель работы заключается в изучении основных управляющих конструкций в языке *Python*, а также методов работы с модулем *numpy* и применении полученных навыков на практике.

Задание

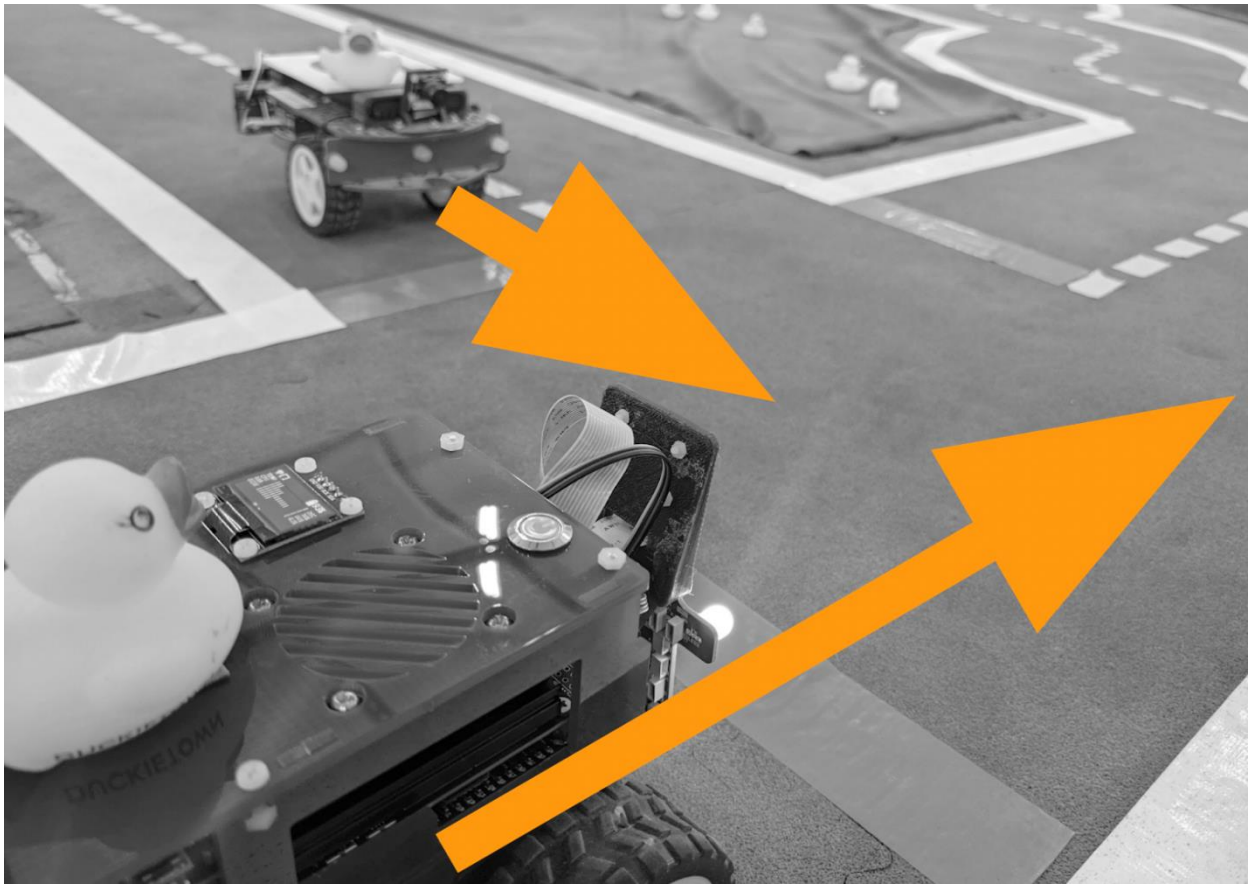
Вариант 1

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. 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

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:

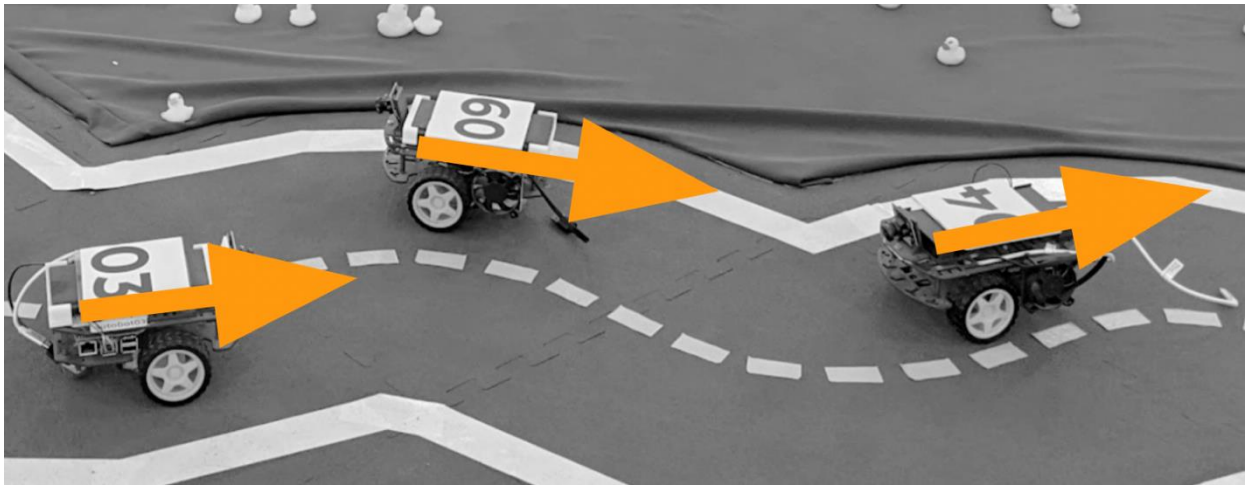
Пример возвращаемого результата:

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 местностности, 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:

Возвращаемый результат:

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.

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)*.

Пример входных аргументов:

Пример возвращаемого результата:

Отчет

В отчете обязательно распишите те методы линейной алгебры и модуля

Основные теоретические положения

Для выполнения лабораторной работы используется модуль *numpy*.

***NumPy* —** это библиотека *Python*, которую применяют для математических вычислений: начиная с базовых функций и заканчивая линейной алгеброй. П

В частности, будут использоваться методы линейной алгебры и для работы с матрицами.

N

N

N

N

N

N

N

N

N

— массив (матрица), образованный списком списков чисел.

a,...)) — добавляет матрицы *b*,… внизу матрицы *a* при условии соответствия размеров.

a,...)) — добавляет матрицы *b*,… справа матрицы *a* при условии соответствия размеров.

— возвращает синус *a*, где *a* — радианы.

— возвращает значение *a*, округленное до *b* знаков после запятой.

Для выполнения поставленных задач подключается модуль *numpy* —

— возвращает результат умножения матрицы *a* на матрицу *b*

— возвращает матрицу, содержащую решение системы линейных уравнений, где

Для выполнения каждой из поставленных задач написана соответствующая функция.

Функция *check_collision* принимает два входных параметра (матрицы движутся дакиботы) и проверяет наличие пересечений траекторий движения этих двух объектов. Для этого функция строит матрицу коэффициентов и матрицу свободный членов, используя входные параметры. Если матрица коэффициентов имеет ранг больше или равный 2, она решает систему уравнений с помощью функции *numpy.linalg.solve* и возвращает координаты точки пересечения, округленные до двух десятичных знаков, в виде кортежа. В противном случае возвращается *None*.

Функция *check_surface* принимает три входных параметра (матрицы определяют ли эти точки плоскость. Для этого функция строит матрицу коэффициентов и матрицу свободных членов, используя входные параметры. Если матрица коэффициентов имеет ранг больше или равный 3, она решает систему уравнений с помощью функции *numpy.linalg.solve* и возвращает коэффициенты уравнения плоскости, округленные до двух десятичных знаков, в виде матрицы *ndarray*. В противном случае возвращается *None*.

Функция *check_rotation* принимает два входных параметра (вектор *ndarray* Для этого функция создает матрицу поворота на основе угла, используя функции матрицей поворота и вектором. Наконец, полученный вектор округляется до двух десятичных знаков, и функция возвращает его значение.

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.			Задача №1
2.		None	Задача №1
3.	*np.array*([1, 2, 3]), *np.array*([3, 5, 2]), *np.array*([-2, 0, 4])	[-0.2 -0.2 3.6]	Задача №2
4.	*np.array*([1, 2, 3]), *np.array*([4, 5, 6]), *np.array*([-7, 8, 9])	None	Задача №2
5.			Задача №3

Выводы

В результате выполнения работы были достигнуты следующие цели:

1. Изучение основных управляющих конструкции в языке *Python*: условные операторы *if-elif-else*. Были освоены основные принципы работы и синтаксис этих конструкций
2. Изучение основных методов работы с модулем *numpy*

3. Применение полученных навыков на практике для закрепления результата

Таким образом, цель работы была успешно достигнута. Изучение основных управляющих конструкций в языке *Python* и их применение на практике позволило усовершенствовать навыки программирования и использования различных инструментов для работы с информацией.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.py

```
import numpy as np

def check_collision(bot1, bot2):
    coefficient_matrix = np.vstack((bot1[:2], bot2[:2]))
    absolute_term_matrix = np.vstack((-bot1[2], -bot2[2]))

    if np.linalg.matrix_rank(coefficient_matrix) >= 2:
        answer = np.linalg.solve(coefficient_matrix,
absolute_term_matrix)
        return (round(answer[0][0], 2), round(answer[1][0], 2))
    else:
        return None

def check_surface(point1, point2, point3):
    coefficient_matrix = np.vstack((np.hstack((point1[:2], 1)),
np.hstack((point2[:2], 1)), np.hstack((point3[:2], 1))))
    absolute_term_matrix = np.vstack((point1[2], point2[2],
point3[2]))

    if np.linalg.matrix_rank(coefficient_matrix) >= 3:
        answer = np.linalg.solve(coefficient_matrix,
absolute_term_matrix)
        return np.array([round(answer[0][0], 2), round(answer[1][0],
2), round(answer[2][0], 2)])
    else:
        return None

def check_rotation(vec, rad):
    rotation_matrix = np.array([[np.cos(rad), -np.sin(rad), 0],
[np.sin(rad), np.cos(rad), 0], [0, 0, 1]])
    vec = rotation_matrix.dot(vec)
    for i in range(3):
        vec[i] = round(vec[i], 2)
    return vec
```