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 _____ Игнатьев К.А.

Преподаватель _____ Иванов Д.В.

Санкт-Петербург

2023

Цель работы

Цель лабораторной работы заключается в решении задач, включающих использование модуля numpy и пакета numpy.linalg для работы с линейной алгеброй. В результате лабораторной работы необходимо разработать и протестировать 3 функции, каждая из которых решает свою задачу.

Задание

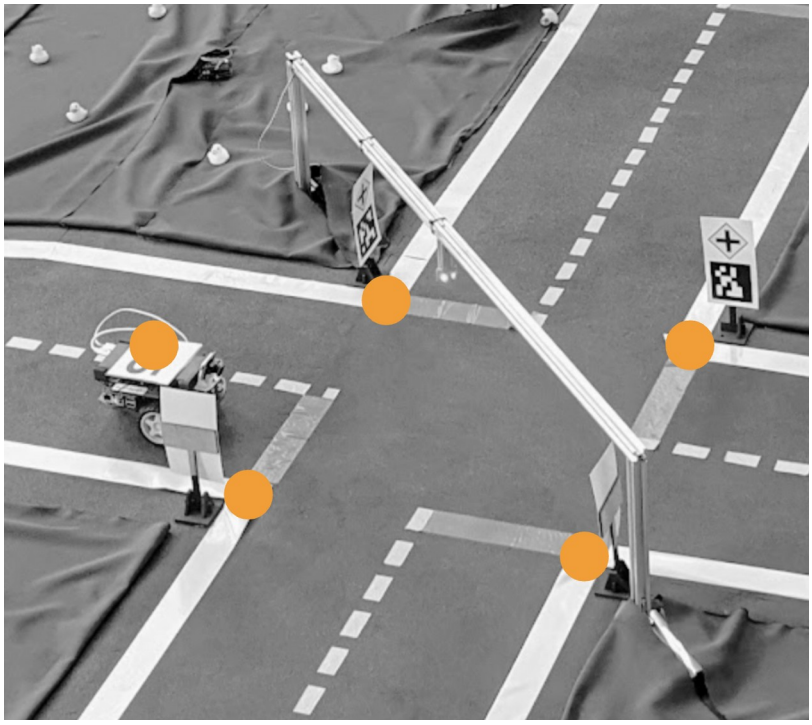
Вариант 2

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.

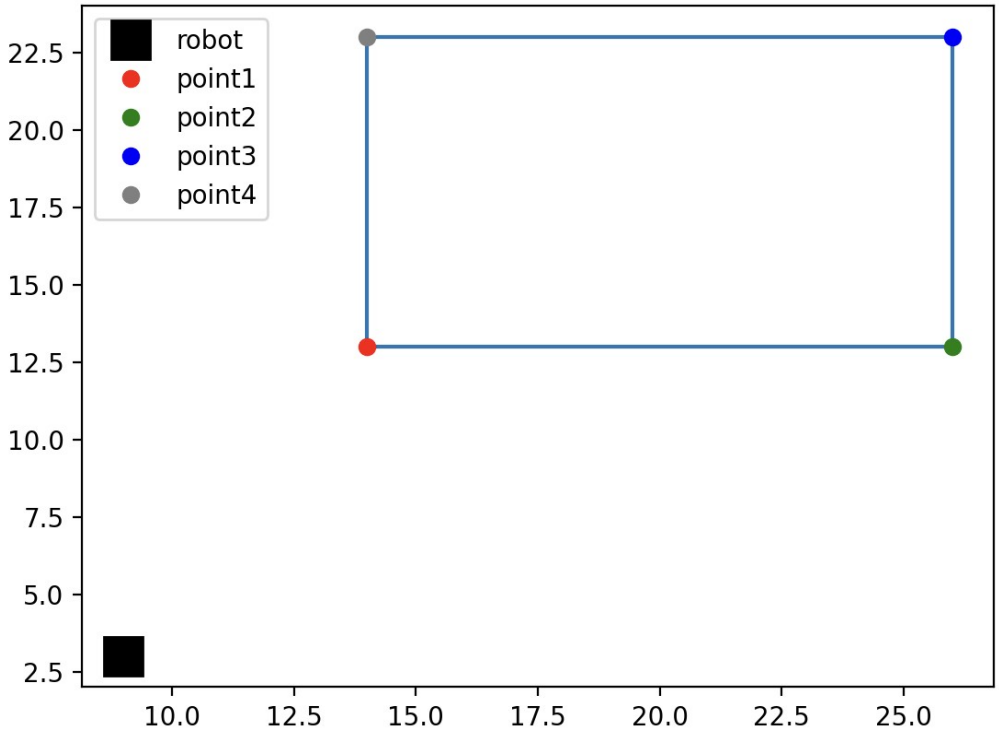
Задача 1. Содержательная постановка задачи

Дакибот приближается к перекрестку. Он знает 4 координаты, соответствующие координатам углов перекрестка (координаты образуют прямоугольник), и свои координаты. По правилам движения дакибот должен остановиться сразу, как только оказывается на перекрестке. Ваша задача -- помочь дакиботу понять, находится ли он на перекрестке (внутри прямоугольника).

Пример ситуации:



Геометрическое представление (вид сверху со схематичным обозначением объектов; перекресток ограничен прямыми линиями; обратите внимание, как пронумерованы точки):



Формальная постановка задачи

Оформите задачу как отдельную функцию: def check_rectangle(robot, point1, point2, point3, point4)

На вход функции подаются: координаты дакибота robot и координаты точек, описывающих перекресток: point1, point2, point3, point4. Точка -- это кортеж из двух целых чисел (x, y).

Функция должна возвращать True, если дакибот на перекрестке, и False, если дакибот вне перекрестка.

Примеры входных аргументов и результатов работы функции:

1. Входные аргументы: (9, 3) (14, 13) (26, 13) (26, 23) (14, 23)

Резлультат: False

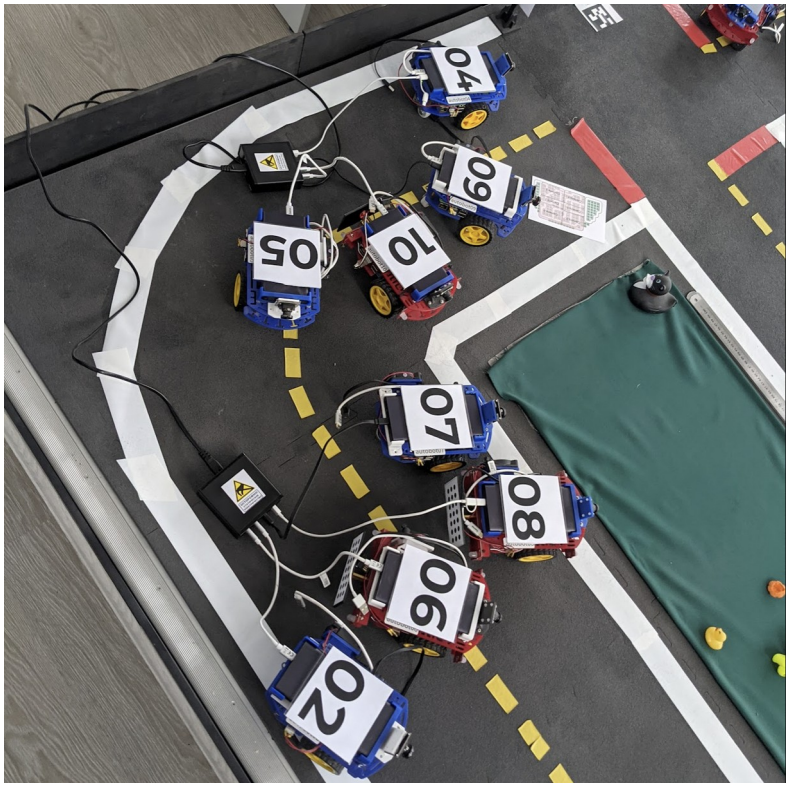
2. Входные аргументы: (5, 8) (0, 3) (12, 3) (12, 16) (0, 16)

Результат: True

Задача 2. Содержательная часть задачи

Несколько дакиботов прибыли на базу, но их корпуса оказались поврежденными. В логах ботов программисты нашли сведения про их траектории движения, которые задаются линейными уравнениями вида: ax+by+c=0. В логах хранятся коэффициенты этих уравнений a, b, c.

Ваша задача -- вывести список номеров ботов (кортежи), которые столкнулись с друг другом (боты нумеруются с нуля, порядок следования коэффициентов уравнений соответствует порядку ботов).



Формальная постановка задачи

Оформите решение в виде отдельной функции check_collision(). На вход функции подается матрица ndarray Nx3 (N -- количество ботов, может быть разным в разных тестах) коэффициентов уравнений траекторий coefficients. Функция возвращает список пар -- номера столкнувшихся ботов (если никто из ботов не столкнулся, возвращается пустой список).

Пример входного аргумента ndarray 4x3 :

[[-1 -4 0]

 [-7 -5 5]

 [1 4 2]

 [-5 2 2]]

Пример выходных данных:

[(0, 1), (0, 3), (1, 0), (1, 2), (1, 3), (2, 1), (2, 3), (3, 0), (3, 1), (3, 2)]

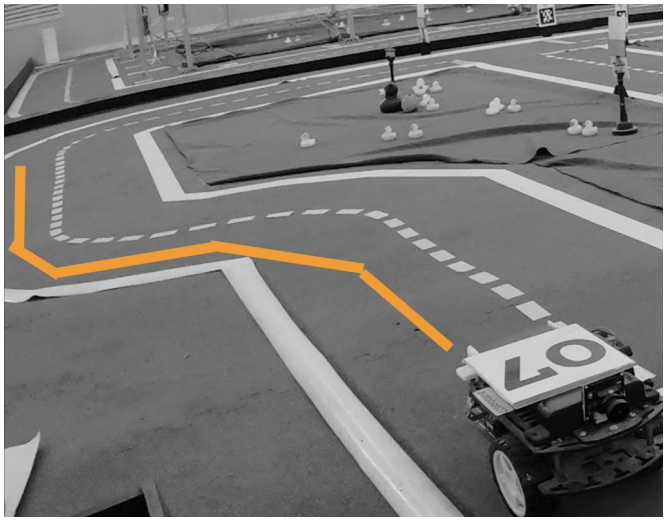
Первая пара в этом списке (0, 1) означает, что столкнулись 0-й и 1-й боты (то есть их траектории имеют общую точку).

В списке отсутствует пара (0, 2), можно сделать вывод, это боты 0-й и 2-й не сталкивались (их траектории НЕ имеют общей точки).

Примечание: помните про ранг матрицы и как от него зависит существование решения системы уравнений. В случае, если ни одного решение не было найдено (например, из-за линейно зависимых векторов), функция должна вернуть пустой список [].

Задача 3. Содержательная часть задачи

При перемещении по дакитауну дакибот должен регулярно отправлять на базу сведения, среди которых есть длина пройденного пути. Дакиботу известна последовательность своих координат (x, y), по которым он проехал. Ваша задача -- помочь дакиботу посчитать длину пути.



Формальная постановка задачи

Оформите задачу как отдельную функцию check_path, на вход которой передается последовательность (список) двумерных точек (пар) points_list. Функция должна возвращать число -- длину пройденного дакиботом пути (выполните округление до 2 знака с помощью round(value, 2)).

Пример входных данных:

[(1.0, 2.0), (2.0, 3.0)]

Пример выходных данных:

1.41

Пример входных данных:

[(2.0, 3.0), (4.0, 5.0)]

Пример выходных данных:

2.83

Основные теоретические положения

Для решения поставленных задач была использована библиотека *numpy*. *NumPy* — это библиотека Python, которую применяют для математических вычислений: начиная с базовых функций и заканчивая линейной алгеброй. Для подключения библиотеки прописана строка *import numpy as np* (в программе для обращения к методам библиотеки используется следующая запись: *np.<название метода>)*

Использованные методы и атрибуты:

1. *np.array* (массив numpy, многомерный массив (*ndarray*, *n-dimensional array*) данных, над которыми можно быстро и эффективно выполнять множество математических, статистических, логических и других операций)
2. *np.ndarray.shape* (кортеж измерений массива)
3. *np.linalg* (функции линейной алгебры *numpy* для обеспечения эффективной низкоуровневой реализации стандартных алгоритмов линейной алгебры)
4. *np.linalg.solve* (Она используется для решения линейных уравнений и нахождения неизвестной переменной или системы линейных скалярных уравнений. В его основе лежит условие: Ax = b)
5. np.sqrt (Возвращает неотрицательный квадратный корень)
6. *np.round* (округление до заданного числа десятичных знаков)

Выполнение работы

Импортируется библиотека *numpy*: *import numpy as np*

В задании требуется оформить каждую из 3 задач в виде отдельной функции согласно условиям.

1. Решение задачи 1:

   def check_crossroad(robot, point1, point2, point3, point4)

   Получает на вход координаты дакибота *robot* и координаты точек, описывающих перекресток: *point1*, *point2*, *point3*, *point4*. Точка – это кортеж из двух целых чисел *(x, y)*.

   Геометрическое представление (вид сверху со схематичным обозначением объектов; перекресток ограничен прямыми линиями) см. на рисунке 1:

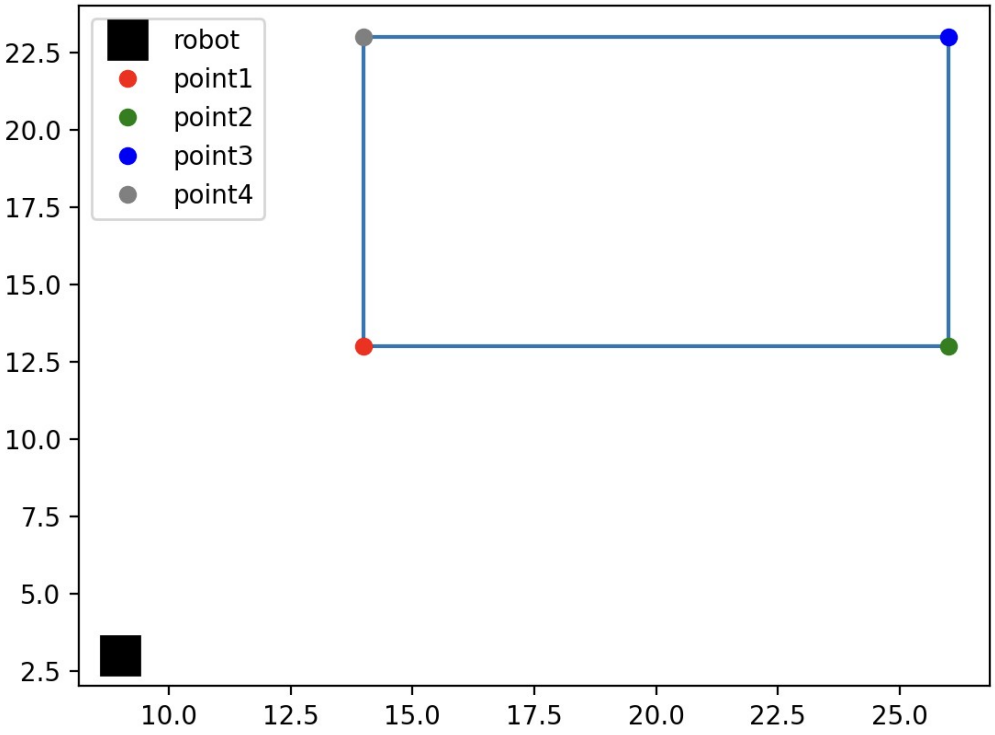


Рисунок 1 – Геометрическое представление

1. Решение задачи 1:

Изначально даны координаты углов перекрестка, являющегося прямоугольником: *point1*, *point2*, *point3*, *point4*. Для решения задачи достаточно проверить, что координаты дакибота *robot(x, y)* меньше минимальных или больше максимальных координат углов перекрестка.

Для этого введем условие: *if (robot[0]<min(point1[0], point2[0], point3[0], point4[0])) or (robot[0]> max(point1[0], point2[0], point3[0], point4[0]))) or ((robot[1]<min(point1[1], point2[1], point3[1], point4[1])) or (robot[1]> max(point1[1], point2[1], point3[1], point4[1])))*. Нужно возвращать значение False, если вышеуказанное условие выполняется, и True в противном случае.

2. Решение задачи 2:

   def check_collision(coefficients)

   На вход функции подается матрица *ndarray Nx3* (*N* -- количество ботов, может быть разным в разных тестах) коэффициентов уравнений траекторий *coefficients*. Функция возвращает список пар – номера столкнувшихся ботов (если никто из ботов не столкнулся, возвращается пустой список).

   Создаётся пустой список *collisions*, в него будут записываться пары столкнувшихся ботов.

   Запускается цикл с ещё одним вложенным:

   for i in range(coefficients.shape[0]):

   for k in range(coefficients.shape[0]): …

   Внутри цикла создаются переменные *LogI* и *LogK*, в которые записываются коэффициенты уравнений. Следом создаются массивы типа ndarray: *A*(для записи первых двух коэффициентов каждого из уравнений системы) и *b*(для записи последних коэффициентов каждого из уравнений системы). Далее, в функции *try* используется функция *np.linalg.solve(A,b)* для решения системы уравнений. В случае нахождения в список *collisions* записывается кортеж из значений *i* и *k*, являющихся номерами ботов. Функция *try* вызывается потому, что, если у системы уравнений нет решений (боты не столкнулись) появляется ошибка *numpy.linalg.LinAlgError: Singular matrix.* В *except* стоит оператор *pass*, так как системы уравнений, не имеющие решения

не нужны. Функция возвращает список кортежей с номерами ботов *collisions*.

Решение задачи 3:

def check_path(points_list)

На вход передается последовательность (список) двумерных точек (пар) *points_list*. Функция должна возвращать число – длину пройденного дакиботом пути (с округлением до 2 знака с помощью *round(value, 2)*).

Внутри функции создается переменная *path*, равная нулю. В нее будут записываться длины отрезков, пройденных дакиботом.

Рассчитывать длину пути мы будем складывая длины отрезков, пройденных дакиботом с помощью формулы: $line = \sqrt{\left(x_0^2 - x_1^2\right) + \left(y_0^2 - y_1^2\right)}$

Создается цикл *for i in range(len(points_list)-1).*

Внутри цикла создаются переменные *a1* и *a2*, в которые записываются разности координат, возведенные в квадрат. Следом к значению переменной *path* прибавляется корень из суммы этих переменных, посчитанный с помощью *np.sqrt*.

Функция возвращает округленное до двух цифр после запятой, с помощью *round* значение переменной *path.*

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.	(9, 3) (14, 13) (26, 13) (26, 23) (14, 23)	False	Задача 1
2.	(5, 8) (0, 3) (12, 3) (12, 16) (0, 16)	True	Задача 1
3.	[[-1 -4 0] [-7 -5 5] [1 4 2] [-5 2 2]]	[(0, 1), (0, 3), (1, 0), (1, 2), (1, 3), (2, 1), (2, 3), (3, 0), (3, 1), (3, 2)]	Задача 2
4.	[(1.0, 2.0), (2.0, 3.0)]	1.41	Задача 3
5.	[(2.0, 3.0), (4.0, 5.0)]	2.83	Задача 3

Выводы

В результате выполнения данной лабораторной работы были достигнуты следующая цели: решение задач, включающих использование модуля *numpy* и пакета *numpy.linalg* для работы с линейной алгеброй, а также разработка и тестирование трех функций, каждая из которых предназначена для решения конкретной задачи.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: lab1.py

```
import numpy as np

def check_crossroad(robot, point1, point2, point3, point4):
     if ((robot[0]<min(point1[0], point2[0], point3[0], point4[0])) or (robot[0]> max(point1[0], point2[0], point3[0], point4[0]))) or ((robot[1]<min(point1[1], point2[1], point3[1], point4[1])) or (robot[1]> max(point1[1], point2[1], point3[1], point4[1]))):
        return False
    else:
        return True

def check_collision(coefficients):
    collisions=[]
    for i in range (coefficients.shape[0]):
        for k in range (coefficients.shape[0]):
            logI=coefficients[i]
            logK=coefficients[k]
            A=np.array([[logI[0], logI[1]], [logK[0], logK[1]]])
            b=np.array([logI[2], logK[2]])
            try:
                np.linalg.solve(A, b)
                collisions.append((i,k))
            except:
                pass
    return collisions

def check_path(points_list):
    path=0
    for i in range(len(points_list)-1):
        a1=(points_list[i][0]-points_list[i+1][0])**2
        a2=(points_list[i][1]-points_list[i+1][1])**2
        path+=np.sqrt(a1+a2)
    return round(path,2)
```