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 _____ Отмахов Д.В.

Преподаватель _____ Иванов Д.В.

Санкт-Петербург

2023

Цель работы.

При помощи основных управляющих конструкций языка Python, библиотеки NumPy реализовать программу, выполняющую различные задачи.

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).

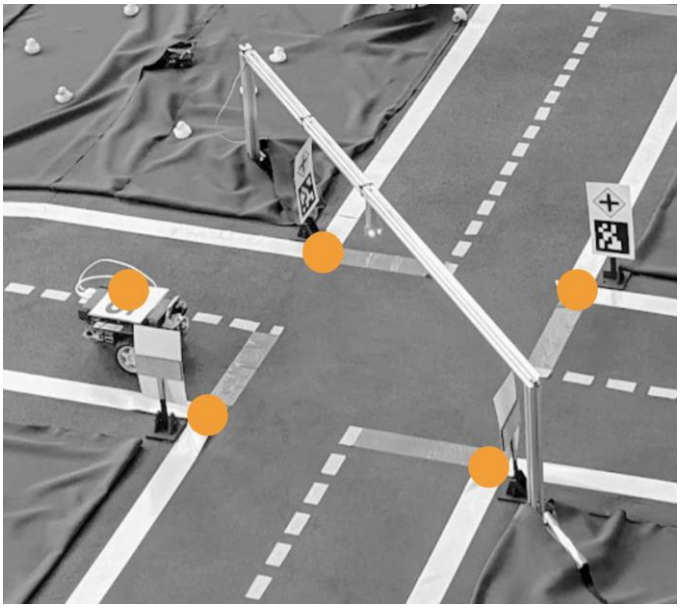

Рисунок 1 – Пример ситуации

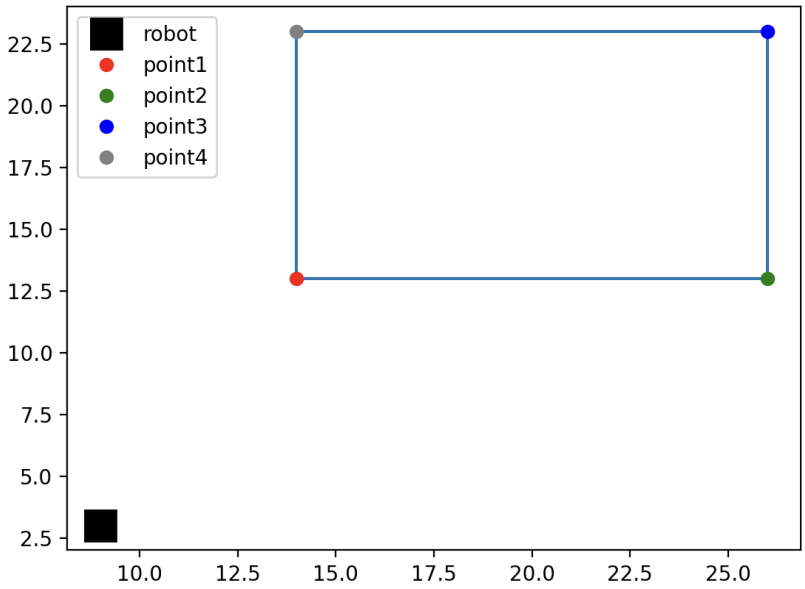

Рисунок 2 – Геометрическое представление (вид сверху со схематичным обозначением объектов; перекресток ограничен прямыми линиями; обратите внимание, как пронумерованы точки)

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.

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).

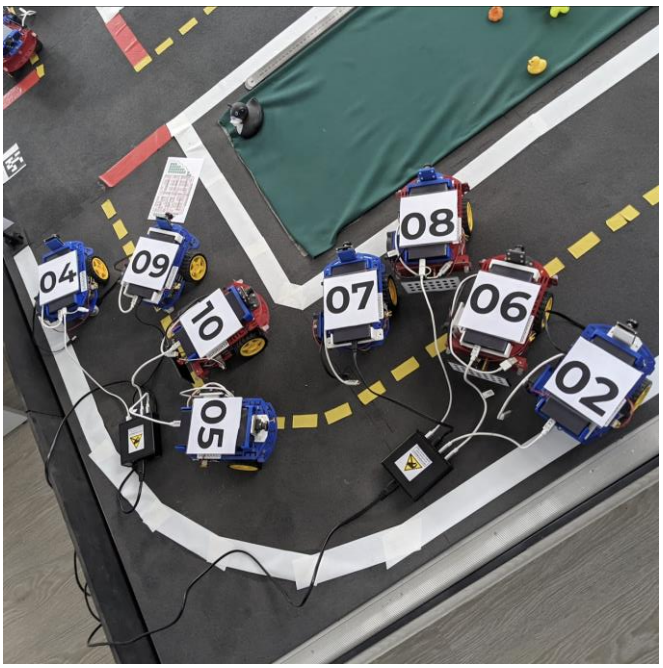

Рисунок 3 – Столкновение дакиботов

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).

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.

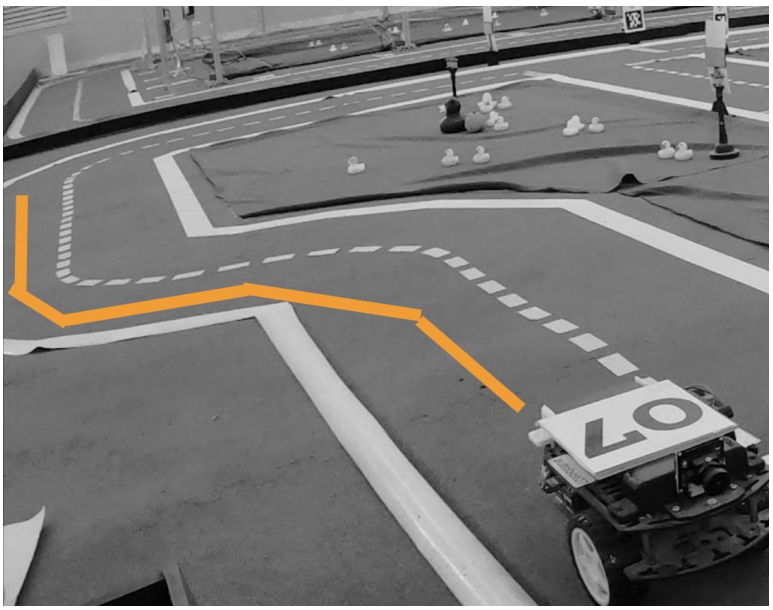

Рисунок 4 – Траектория движения дакибота

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)*).

Отчет

В отчете обязательно распишите те методы линейной алгебры и модуля numpy, которые вы использовали при решении задач.

Выполнение работы.

Данная программа написана на языке Python. В ней реализованы 3 функции, выполняющие 3 поставленные задачи для управления дакиботами.

Описание функций:

1) Функция *check_crossroad(robot, point1, point2, point3, point4)* принимает на вход координаты дакибота (*robot*) и координаты точек, описывающих перекресток (*point1*, *point2*, *point3*, *point4*). Функция проверяет нахождение дакибота на перекрестке путем сравнения координат дакибота с координатами перекрестка и возвращает *True*, если дакибот находится на перекрестке, или *False*, если дакибот вне перекрестка.

2) Функция *check_collision(coefficients)* принимает на вход матрицу *ndarray* Nx3 (N – количество дакиботов) коэффициентов уравнений траекторий движения дакиботов (*coefficients*). При помощи вложенных циклов for в функции рассматриваются все возможные пары дакиботов. Коэффициенты этих пар дакиботов записываются в переменные *a1*, *b1*, *c1* и *a2*, *b2*, *c2* (первого и второго дакибота пары, соответственно). При помощи метода *matrix_rank* модуля *numpy* пакета *linalg*, который возвращает ранг матрицы *[[a1, b1], [a2, b2]]*, проверяется столкновение дакиботов. Далее при помощи условного оператора *if*, полученное значение сравнивается с необходимым для пересечения траекторий движения дакиботов, а именно 2. Если ранг матрицы равен двум, то есть произошло столкновение дакиботов, то их номера записываются парой (кортежем) в список *pairs*. Функции возвращает отсортированный список *pairs* – список пар, столкнувшихся дакиботов.

3) Функция *check_path(points_list)* принимает на вход список двумерных точек (пар). При помощи цикла *for* она вычисляет длину пройденного дакиботом пути между двумя соседними точками и прибавляет это значение к длине суммарного пути (*path_length*). Функция возвращает значение *path_length* (длину пройденного дакиботом пути), округленное до 2 знаков при помощи функции *round*. Для вычисления пути был использована функция sqrt модуля math.

Разработанный программный код см. в приложении А.

Тестирование.

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

№	Входные данные	Выходные данные	Комментарий
1.	(9, 3) (14, 13) (26, 13) (26, 23) (14, 23)	False	Функция *check_crossroad*
2.	(5, 8) (0, 3) (12, 3) (12, 16) (0, 16)	True	Функция *check_crossroad*
3.	[[-1 -4 0] [-7 -5 5]	[(0, 1), (0, 3), (1, 0), (1, 2), (1, 3), (2, 1), (2, 3), (3, 0), (3, 1), (3, 2)]	Функция *check_collision*

	[1 4 2] [-5 2 2]]		
4.	[(1.0, 2.0), (2.0, 3.0)]	1.41	Функция *check_path*
5.	[(2.0, 3.0), (4.0, 5.0)]	2.83	Функция *check_path*

Выводы.

В рамках выполнения лабораторной работы при помощи основных управляющих конструкций языка Python, библиотеки NumPy была реализована программа, выполняющая различные задачи.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.py

```
import numpy as np
import math


def check_crossroad(robot, point1, point2, point3, point4):
    return point1[0] <= robot[0] <= point3[0] and point1[1] <= robot[1] <= point3[1]


def check_collision(coefficients):
    pairs = []
    for i in range(len(coefficients)):
        for j in range(len(coefficients)):
            a1, b1, c1 = coefficients[i]
            a2, b2, c2 = coefficients[j]
            if np.linalg.matrix_rank(np.array([[a1, b1], [a2, b2]])) == 2:
                pairs.append((i, j))
    return sorted(pairs)


def check_path(points_list):
    path_length = 0
    for i in range(len(points_list) - 1):
        x1, y1 = points_list[i]
        x2, y2 = points_list[i + 1]
        path_length = path_length + math.sqrt((x2 - x1) ** 2 + (y2 - y1) ** 2)
    return round(path_length, 2)
```