**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 Python**

| Студент(ка) гр. 3383 |  | Логинова А. Ю. |
| --- | --- | --- |
| Преподаватель |  | Иванов Д. В. |

Санкт-Петербург

2023

# Цель работы.

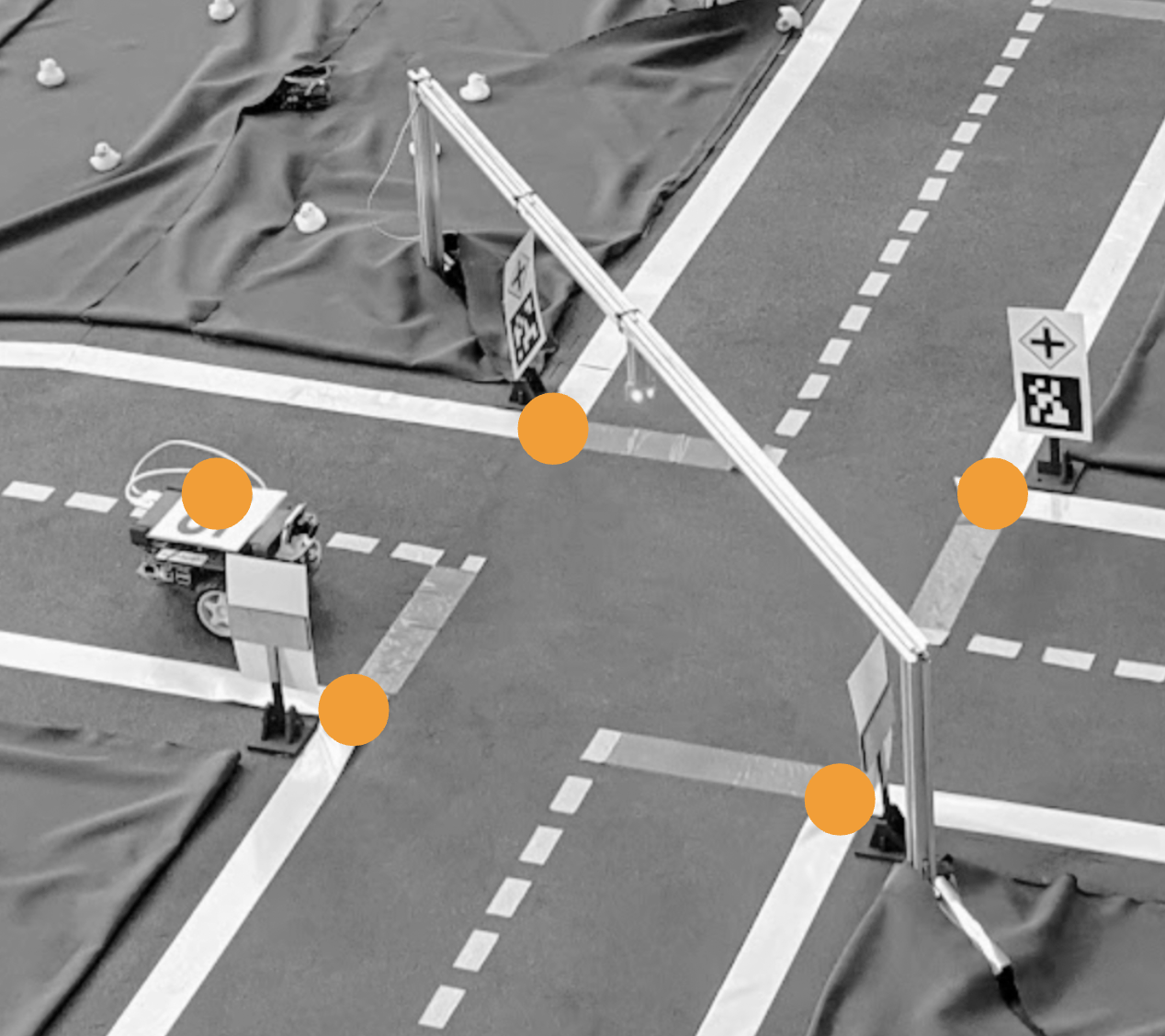
Применить знания основных конструкций Python в разработке, научиться работать с модулем Numpy.

# Задание.

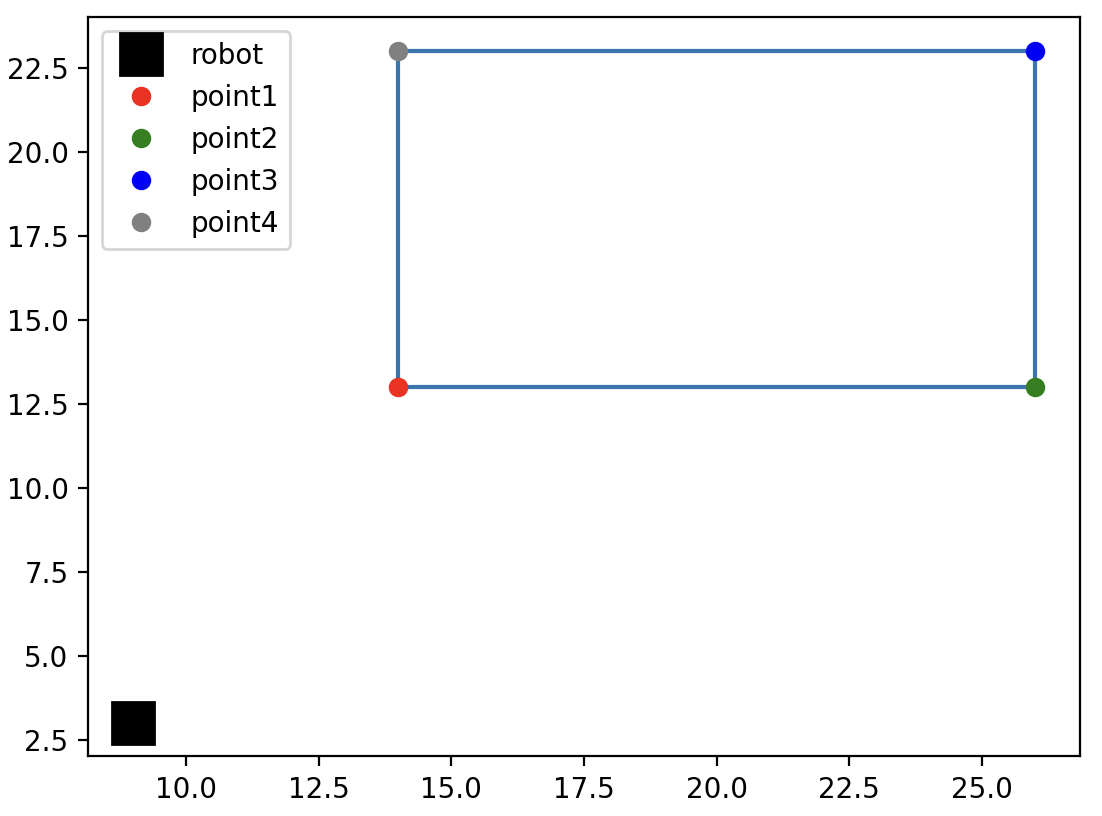
## Задача 1. Содержательная постановка задачи

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\_crossroad(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)

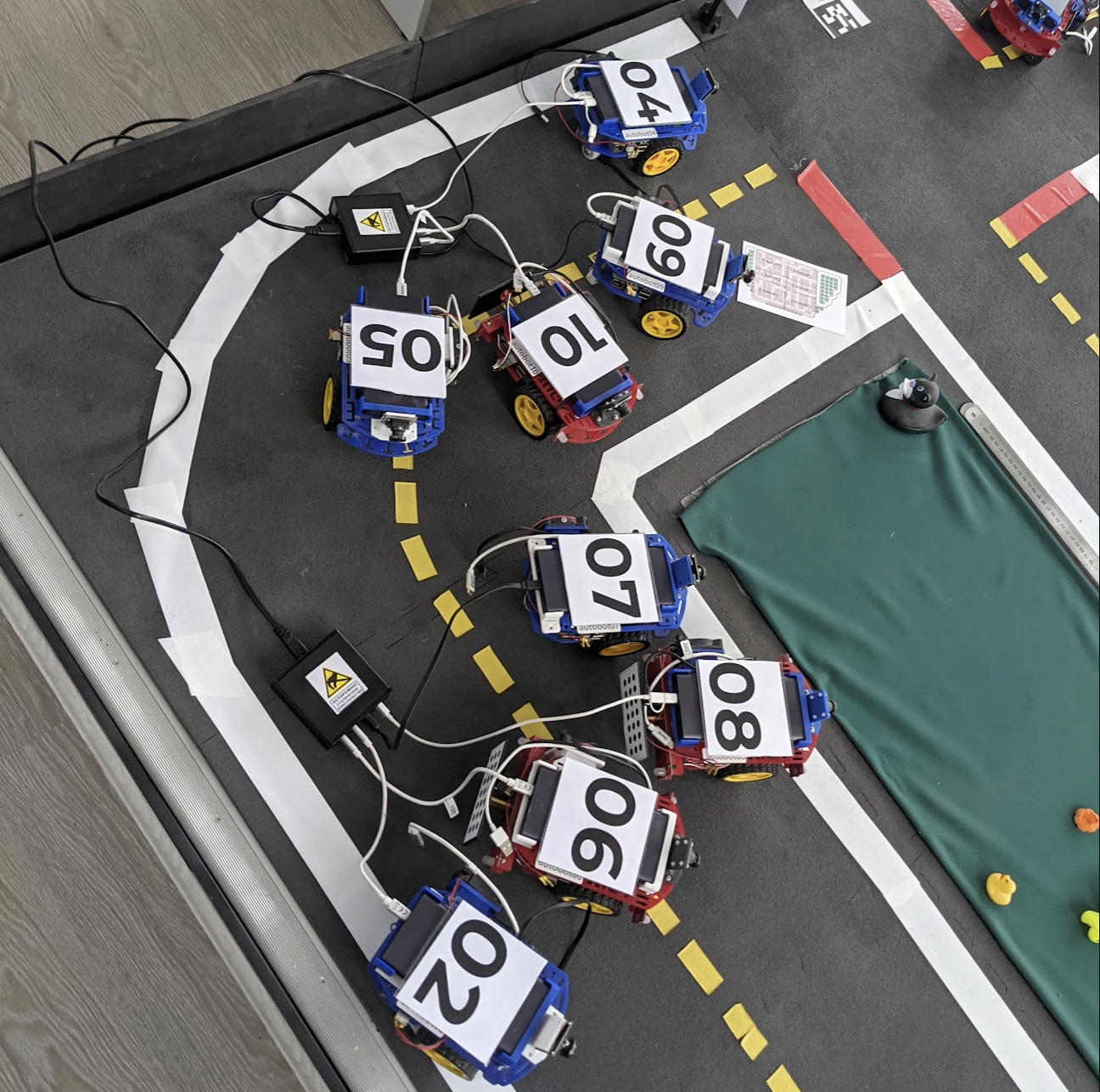
Результат: True

## 

## 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:

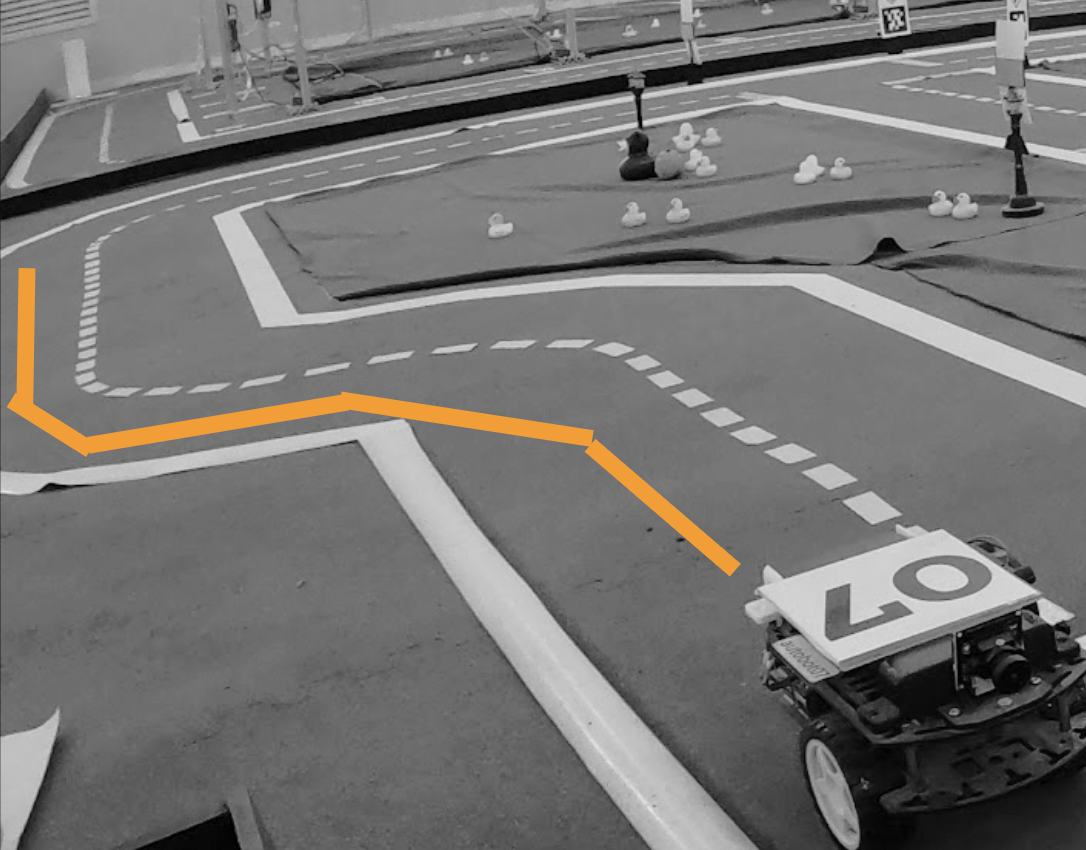
[(0, 1), (0, 3), (1, 0), (1, 2), (1, 3), (2, 1), (2, 3), (3, 0), (3, 1), (3, 2)]

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

# 

# 

# Выполнение работы.

На вход функции check\_crossroad(robot, point1, point2, point3, point4) подаются 5 точек с двумя координатами в виде кортежа. Точка robot - координаты точки, проверку вхождения в прямоугольник которого надо проверить. Точки point - координаты прямоугольника.

В функции считается длина сторон прямоугольникам путем выcчитывания длины вектора функцией vector\_length(), которая принимает на вход 2 точки и с помощью вычета векторов и метода norm() из Numpy.linalg возвращает значение длины. Таким образом, ab, bc, cd, da - стороны прямоугольника. Так же считается и расстояние от точки робота до каждой вершины прямоугольника: ar, br, cr, dr. Функция считает площадь треугольников, которые образуют эти 4 точки. Если сумма площадей треугольников совпадает с площадью прямоугольника, тогда точка находится в прямоугольнике. Площадь треугольников вычислялась с помощью дополнительной функции triangle\_area(), в которой с помощью формула Герона по 3-м точкам находится площадь любого треугольника, для этого задействуется модуль math. Данная функция возвращает число с плавающей точкой, поэтому сумму площадей треугольников необходимо округлить с помощью метода round(). С помощью условного оператора if была осуществлена проверка площадей, возвращается True/False.

На вход функции check\_path() подается матрица 1xn, где каждая строка - это координаты точки, в которой побывал робот. Расстояние пройденного пути считается функцией vector\_length(), описанной выше, и суммируется в переменную path, значение которой округляется с помощью метода round(). Переменная path возвращается.

На вход функции check\_collision() подается матрица nx3, где каждая строка - линейное уравнения прямой, по которой двигался робот. Необходимо вернуть пару номеров столкнувшихся роботов. Чтобы узнать, столкнулись роботы или нет, надо решить система уравнений, с помощью Numpy это можно сделать в матричном виде. Самый правый столбец переданной на вход матрицы - это свободные члены данных прямых, его будет удобно отделить от неизвестных, поменяв всем значениям знак. Два левых столбца это коэффициенты при неизвестных, поэтому они записываются в переменную unknown, правый столбец в переменную solution (меняя знак).

Поиск столкнувшихся ботов осуществлен вложенным циклом. Внешний цикл берет 1-ю строку из unknown, внутренний цикл берет 1-ю строку из unknown. Далее проверяется наличие решений с помощью метода solve() из Numpy.linalg. Поскольку данный метод, при отсутствии решений у системы, будет выдавать ошибку, использовалась конструкция try, catch, чтобы вместо объявления об ошибке, возвращался пустой список решений. Если решение есть, записываются индексы текущей итерации в массив collisions. После окончания работы цикла он возвращается.

# Тестирование.

| (9, 3) (14, 13) (26, 13) (26, 23) (14, 23)  [[-1 -4 0]  [-7 -5 5]  [ 1 4 2]  [-5 2 2]]  [(1.0, 2.0), (2.0, 3.0)] | False  [(0, 1), (0, 3), (1, 0), (1, 2), (1, 3), (2, 1), (2, 3), (3, 0), (3, 1), (3, 2)]  1.41 |
| --- | --- |
| (5, 8) (0, 3) (12, 3) (12, 16) (0, 16)  [[ 5 5 10]  [ 8 -2 2]  [ 2 -10 5]  [ -7 -5 10]  [-10 -2 2]  [ -8 5 1]  [ -6 -4 8]  [ 3 3 5]]  [(2.0, 3.0), (4.0, 5.0)] | True  [(0, 1), (0, 2), (0, 3), (0, 4), (0, 5), (0, 6), (0, 7), (1, 0), (1, 2), (1, 3), (1, 4), (1, 5), (1, 6), (1, 7), (2, 0), (2, 1), (2, 3), (2, 4), (2, 5), (2, 6), (2, 7), (3, 0), (3, 1), (3, 2), (3, 4), (3, 5), (3, 6), (3, 7), (4, 0), (4, 1), (4, 2), (4, 3), (4, 5), (4, 6), (4, 7), (5, 0), (5, 1), (5, 2), (5, 3), (5, 4), (5, 6), (5, 7), (6, 0), (6, 1), (6, 2), (6, 3), (6, 4), (6, 5), (6, 7), (7, 0), (7, 1), (7, 2), (7, 3), (7, 4), (7, 5), (7, 6)]  2.83 |

# Выводы.

Были изучены основные конструкции Python, применены различные методы модуля Numpy. Была разработана программа для подсчета пути, для выявления столкновений роботов.

ПРИЛОЖЕНИЕ А

## Исходный код программы.

Название файла: main.py

import math

import numpy

def vector\_length(point1, point2):

return numpy.linalg.norm(numpy.array((point1)) - numpy.array((point2)))

def triangle\_area(side1, side2, side3):

s = (side1 + side2 + side3) / 2

area = math.sqrt(s \* (s - side1) \* (s - side2) \* (s - side3))

return area

def check\_crossroad(robot, point1, point2, point3, point4):

ab = vector\_length(point1, point2)

bc = vector\_length(point2, point3)

cd = ab

da = bc

rect\_area = ab \* bc

ar = vector\_length(robot, point1)

br = vector\_length(robot, point2)

cr = vector\_length(robot, point3)

dr = vector\_length(robot, point4)

arb = triangle\_area(ar, br, ab)

brc = triangle\_area(br, cr, bc)

crd = triangle\_area(cr, dr, cd)

dra = triangle\_area(dr, ar, da)

tri\_area = round(arb + brc + crd + dra)

if tri\_area == rect\_area:

return True

else:

return False

def check\_collision(coefficients):

unknown = numpy.array([(line[0], line[1]) for line in coefficients])

solution = numpy.array([(-elem[2]) for elem in coefficients])

collisions = []

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

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

try:

numpy.linalg.solve((unknown[i], unknown[k]), (solution[i], solution[k]))

found = True

except numpy.linalg.LinAlgError:

found = False

if found:

collisions.append((i, k))

return collisions

def check\_path(points\_list):

path = 0

for i in range(len(points\_list) - 1):

path += vector\_length(points\_list[i], points\_list[i+1])

return round(path, 2)