Второй отборочный этап

Второй отборочный этап проводится в командном формате в сети интернет, работы оцениваются автоматически средствами системы онлайн-тестирования. Продолжительность второго этапа составляет 52 дня. Задачи по информатике носят междисциплинарный характер и помогают отработать те навыки, которые потребуются для решения командной задачи заключительного этапа.

Участники не были ограничены в выборе языка программирования для решения задач.

Объем и сложность задач этого этапа подобраны таким образом, чтобы решение всех задач одним человеком было маловероятно. Это призвано обеспечить включение командной работы и распределения обязанностей. Решение каждой задачи дает определенное количество баллов. Баллы зачисляются в полном объеме за правильное решение задачи. Также существуют задачи, где допускается частичное решение. В данном этапе можно получить суммарно от 0 до 55 баллов.

Задачи по программированию выкладывались тремя партиями: в начале второго этапа, через две недели после начала и через пять недель после начала. Команды могут выполнять задачи в любом порядке. Задачи допускают неограниченное число попыток сдать решение.

Задачи второго этапа

Задачи по информатике

Задача II.1.1.1. Цилиндры (5 баллов)

Робототехническое устройство, собранное по дифференциальной схеме и с заданным диаметром колес, движется по прямой с постоянной скоростью. На устройстве установлен ультразвуковой датчик расстояния, направленный влево перпендикулярно направлению движения.

Слева от робототехнического устройства установлены в ряд цилиндры разного диаметра d_i. Цилиндры расположены таким образом, что все их центры находятся на одной прямой. Высота цилиндров много выше высоты, на которой установлен датчик расстояния. Цилиндры могут располагаться как вплотную друг к другу, так и с зазором.

Во время движения датчик расстояния, устроенный таким образом, что он имеет конус направленности $\alpha = 1''$ и возвращает минимальное значение из заданного диапазона или максимально возможное в случае, если отсутствуют какие-либо объекты в непосредственной видимости датчика. Известно, что показания ультразвукового **датчика идеальны**, в них всегда присутствует высокая доля помех. Частота опроса датчика — 10 Гц.

Необходимо найти цилиндр максимального диаметра. Известно, что радиус колес

равен 0.09 м.

При движении робота не гарантируется, что он движется параллельно прямой, на которой установлены цилиндры. Гарантируется, что робот движется так, что во время движения фиксирует датчиком все установленные цилиндры таким образом, что было сделано не менее 3 изменений для каждого цилиндра.

Формат входных данных

Первая строка входных данных содержит одно целое число — N, где:

- N — количество измерений ($3 \leqslant N \leqslant 1000$).

Далее идут N строк, содержащие 2 вещественных числа через пробел — Enc, S, где:

- Enc — среднее арифметическое показаний энкодеров левого и правого колеса в градусах ($0 \leqslant Enc \leqslant 10^6$);
- S — показание датчика расстояния в мм ($10 \leqslant S \leqslant 3700$).

Формат выходных данных

Одна строка, содержащая одно целое число — номер цилиндра максимального диаметра считая по ходу движения.

Примеры

Примеры входных данных и ответов к ним можно найти по данной ссылке.

Решение

Декомпозиция решения задачи:

1. Определить количество цилиндров в наборе данных и принадлежащие им точки на окружности
2. Вычислить диаметр каждого цилиндра по трем точкам на окружности
3. Определить номер цилиндра с максимальным диаметром

Для начала получим угол обзора камеры в десятичном формате из угловых значений ($d^\circ m' s''$) в десятичный формат (градусы) по формуле (II.1.1):

$$degrees = d + \frac{m}{60} + \frac{s}{3600} \tag{II.1.1}$$

В нашем случае получается $1'' = 0.0002778$ градусов, т.е. угол обзора незначительный и можно его принять за “прямую”, направленную перпендикулярно влево по ходу движения робота.

Для примера возьмем наборы данных N1 и N2. Посмотрим на данные в виде графиков (рисунки II.1.1 и II.1.2):

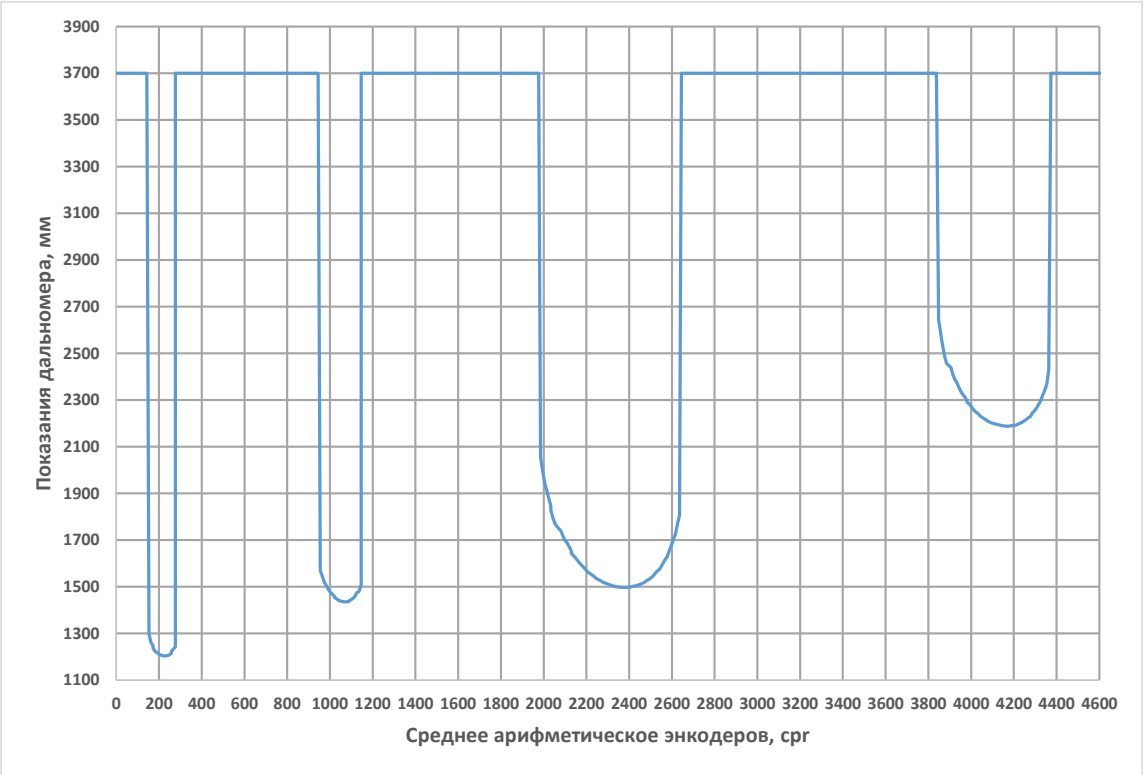

Рис. II.1.1: Набор данных N1

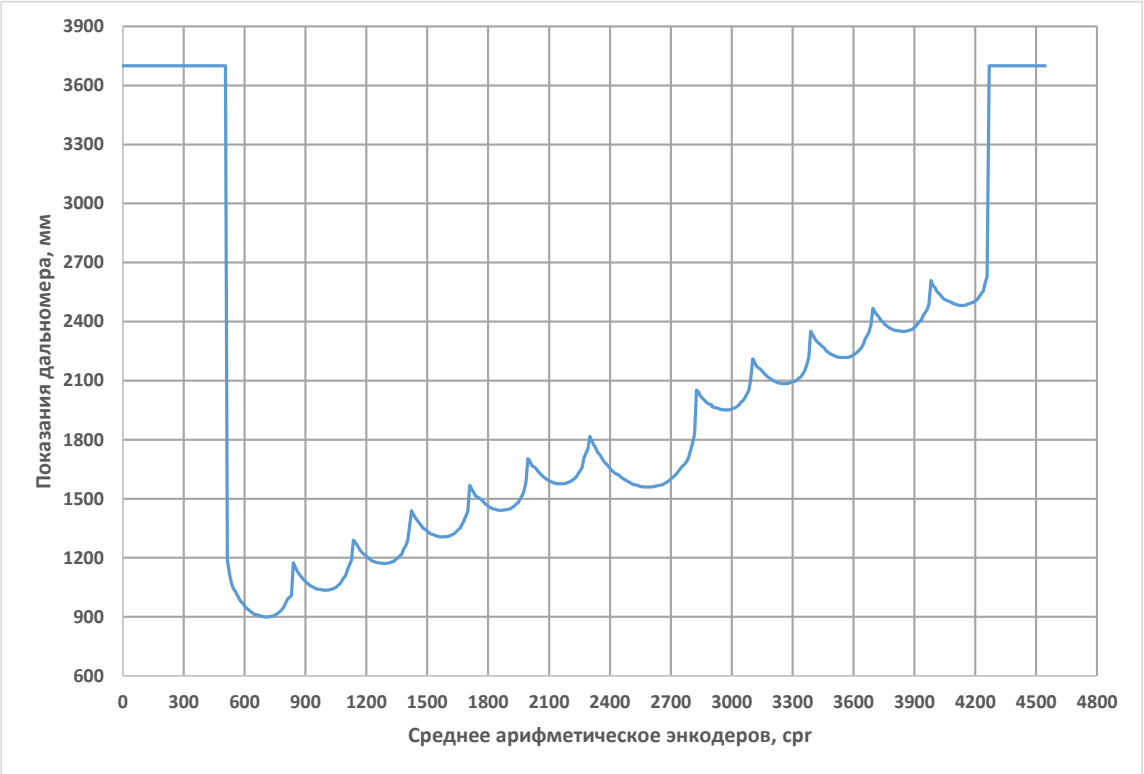

Рис. II.1.2: Набор данных N2

По графикам видно:

1. Искомые цилиндры могут располагаться очень близко друг к другу или между ними может быть какое-либо расстояние.
2. Максимальное расстояние в наборе данных, возвращаемое датчиком расстояния - 3700 мм, т.е. в этом случае отсутствуют какие-либо объекты.
3. Робот может двигаться как параллельно цилиндрам, так и по диагонали.

Декомпозируем задачу определения каждого цилиндра:

1. Определить начальную точку цилиндра (M_1).
2. Определить минимальное показание датчика расстояния относительно начальной точки цилиндра (M_2).
3. Определить конечную точку цилиндра.
4. Присвоить цилиндру номер (M_3).

Результаты определения точек каждого цилиндра для набора данных N1 показан на рисунке II.1.3:

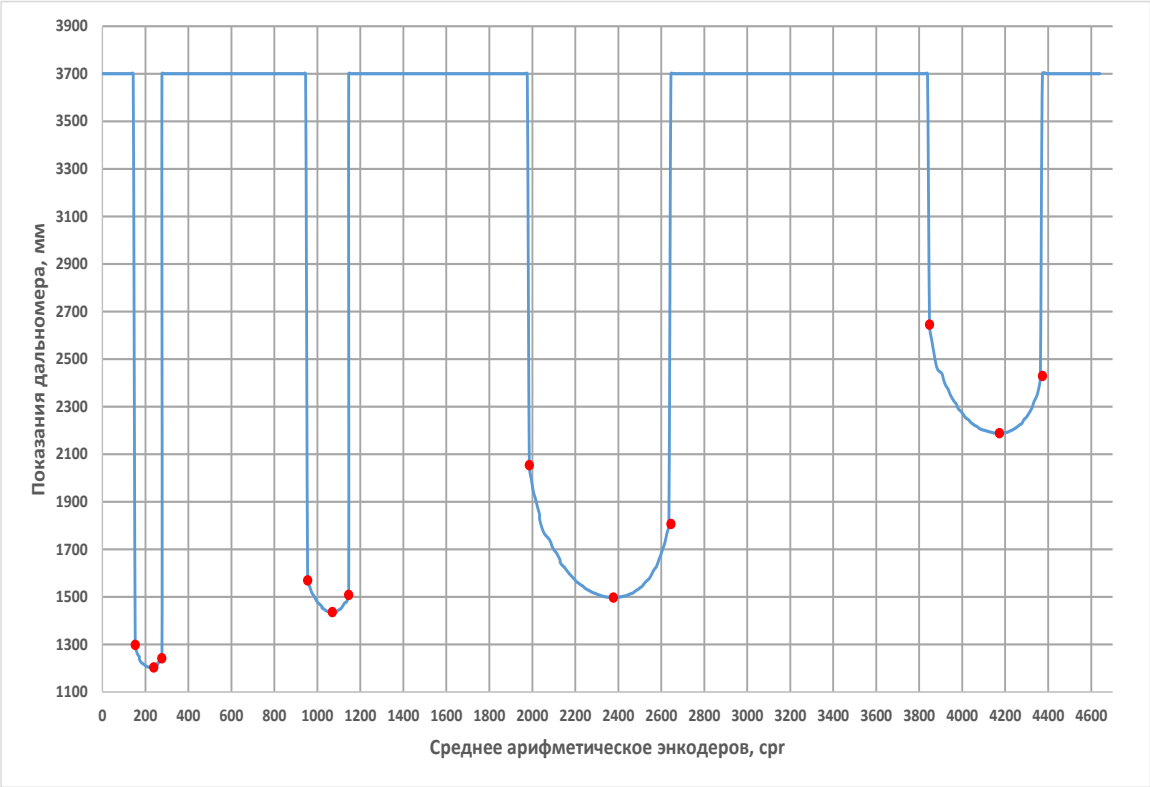

Рис. II.1.3: Крайние точки цилиндров из набора данных N1

Координаты точек указаны в таблице II.1.1:

Далее, по полученным 3 точкам каждого цилиндра (M_1, M_2, M_3) определяем их диаметры. Для этого сначала находим центры цилиндров. Схема нахождения центра цилиндра по трем известным точкам показана на рис. В примере использованы точки цилиндра N2 из набора данных N1.

Цилиндр	Точка	X	Y
1	M_1	152,634042	1298,516512
	M_2	229,025706	1203,757882
	M_3	276,769403	1242,196202
2	M_1	954,744156	1569,562435
	M_2	1069,330793	1435,830474
	M_3	1145,721379	1508,241296
3	M_1	1986,027389	2053,845644
	M_2	2377,531502	1497,469187
	M_3	2644,901633	1806,757331
4	M_1	3848,065298	2644,686222
	M_2	4172,728161	2188,261747
	M_3	4373,256726	2429,352999

Таблица II.1.1: Координаты точек цилиндров из набора данных N1

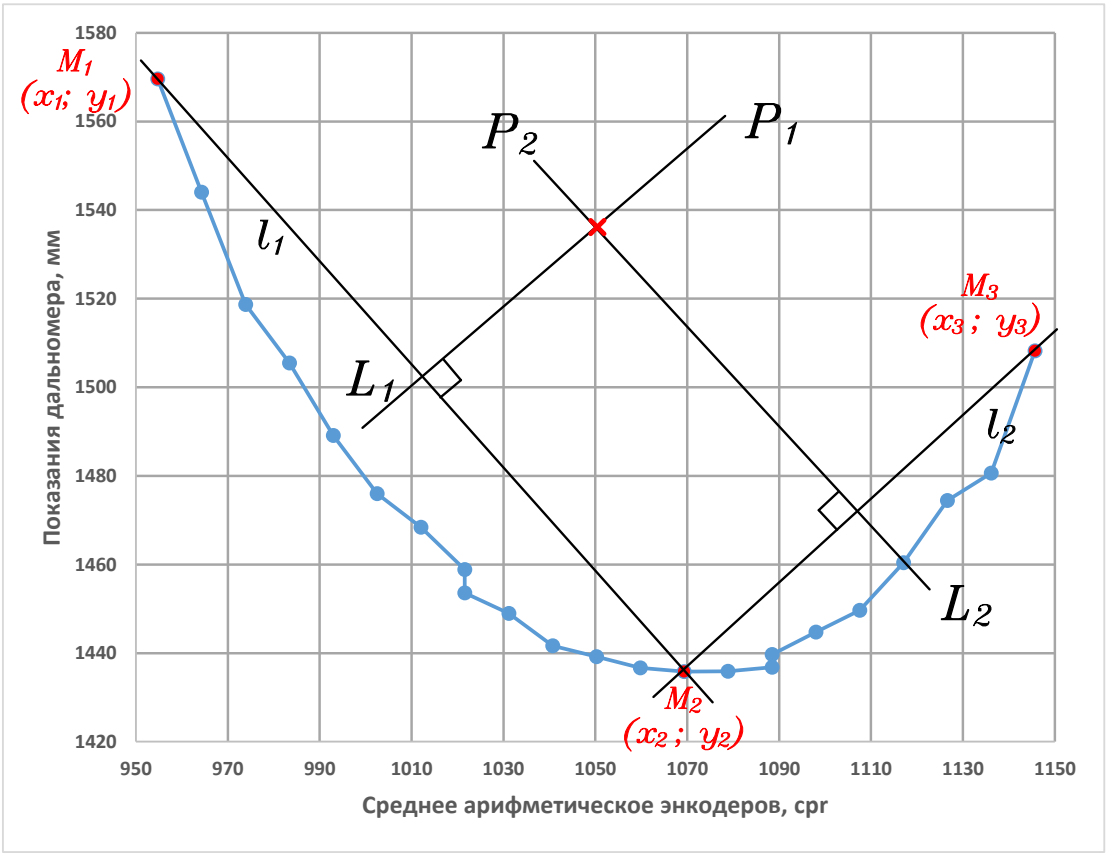

Рис. II.1.4: Схема нахождения центра цилиндра по 3 известным точкам

Уравнение прямой, проходящей через две точки, имеет вид:

$$\frac{y - y_1}{y_2 - y_1} = \frac{x - x_1}{x_2 - x_1}$$

В этом случае угловой коэффициент k определятся по формуле:

$$k = \frac{y_2 - y_1}{x_2 - x_1};$$

Найдем коэффициенты k и b для прямых l_1 и l_2 по формулам (II.1.2) и (II.1.2), (II.1.2) и (II.1.2):

$$\begin{aligned} k_{l_1} &= \frac{y_2 - y_1}{x_2 - x_1} \\ b_{l_1} &= y_2 - k_{l_1} \cdot x_2 \\ k_{l_2} &= \frac{y_3 - y_2}{x_3 - x_2} \\ b_{l_2} &= y_3 - k_{l_2} \cdot x_3 \end{aligned} \tag{II.1.2}$$

Центр цилиндра находится на пересечении двух перпендикулярных прямых P_1 и P_2, проходящих через середины отрезков M_1M_2 и M_2M_3.

Найдем середины отрезков M_1M_2 и M_2M_3 по формулам (II.1.3) и (II.1.3).

$$\begin{aligned} L_1 &= \left(\frac{x_1 + x_2}{2}; \frac{y_1 + y_2}{2} \right) \\ L_2 &= \left(\frac{x_2 + x_3}{2}; \frac{y_2 + y_3}{2} \right) \end{aligned} \tag{II.1.3}$$

Прямая, перпендикулярная к линии с коэффициентом наклона k имеет коэффициент наклона: $-1/k$, значит уравнения прямых P_1 и P_2, перпендикулярных l_1 и l_2, соответственно, запишем следующим образом: для P_1 – по формулам (II.1.4) и (II.1.4), для P_2 - по формулам (II.1.4) и (II.1.4):

$$\begin{aligned} k_{P_1} &= -\frac{1}{k_{l_1}} \quad \Leftrightarrow \quad k_{P_1} = -\frac{x_2 - x_1}{y_2 - y_1} = \frac{x_1 - x_2}{y_2 - y_1} \\ b_{P_1} &= y_{L_1} - k_{P_1} \cdot x_{L_1} \\ k_{P_2} &= -\frac{1}{k_{l_2}} \quad \Leftrightarrow \quad k_{P_2} = -\frac{x_3 - x_2}{y_3 - y_2} = \frac{x_2 - x_3}{y_3 - y_2} \\ b_{P_2} &= y_{L_2} - k_{P_2} \cdot x_{L_2} \end{aligned} \tag{II.1.4}$$

Сейчас мы уже можем найти координаты точки центра цилиндра (или точку пересечения прямых P_1 и P_2) по формулам (II.1.5) и (II.1.5):

$$\begin{aligned} x &= \frac{b_{P_1} - b_{P_2}}{k_{P_2} - k_{P_1}} \\ y &= k_{P_1} \cdot x + b_{P_1} \end{aligned} \tag{II.1.5}$$

Для нашего примера (цилиндр N2 набора данных N1) координаты центра цилиндра равны: $M_0(1051.406; 1528.247)$

Диаметр цилиндра вычислим по среднему расстоянию от центра цилиндра до каждой из известных точек M_i. Расстояние между двумя точками находим по формуле:

$$r = \sqrt{(M_{ix} - M_{0x})^2 + (M_{iy} - M_{0y})^2}$$

В качестве ответа выводим номер цилиндра с наибольшим диаметром.

Результаты вычислений по наборам данных N1 и N2 показаны на рис. II.1.5 и II.1.6

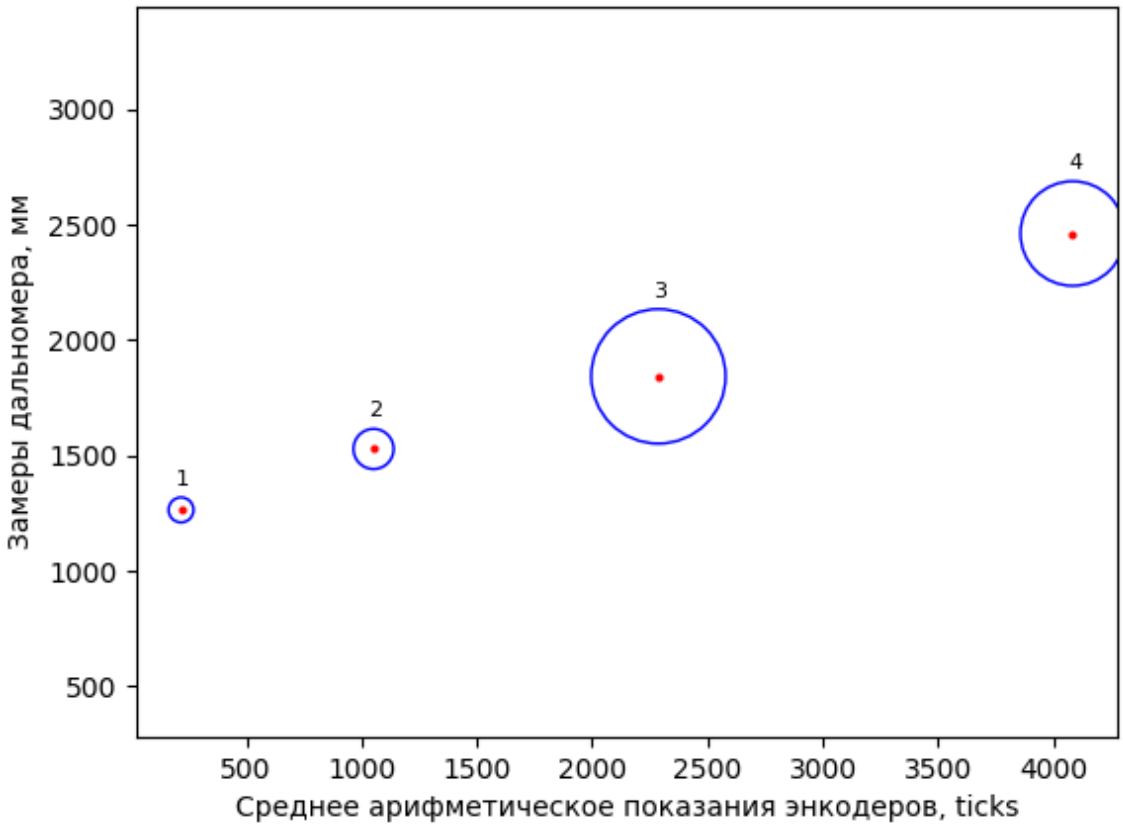

Рис. II.1.5: Цилиндры из набора данных N1

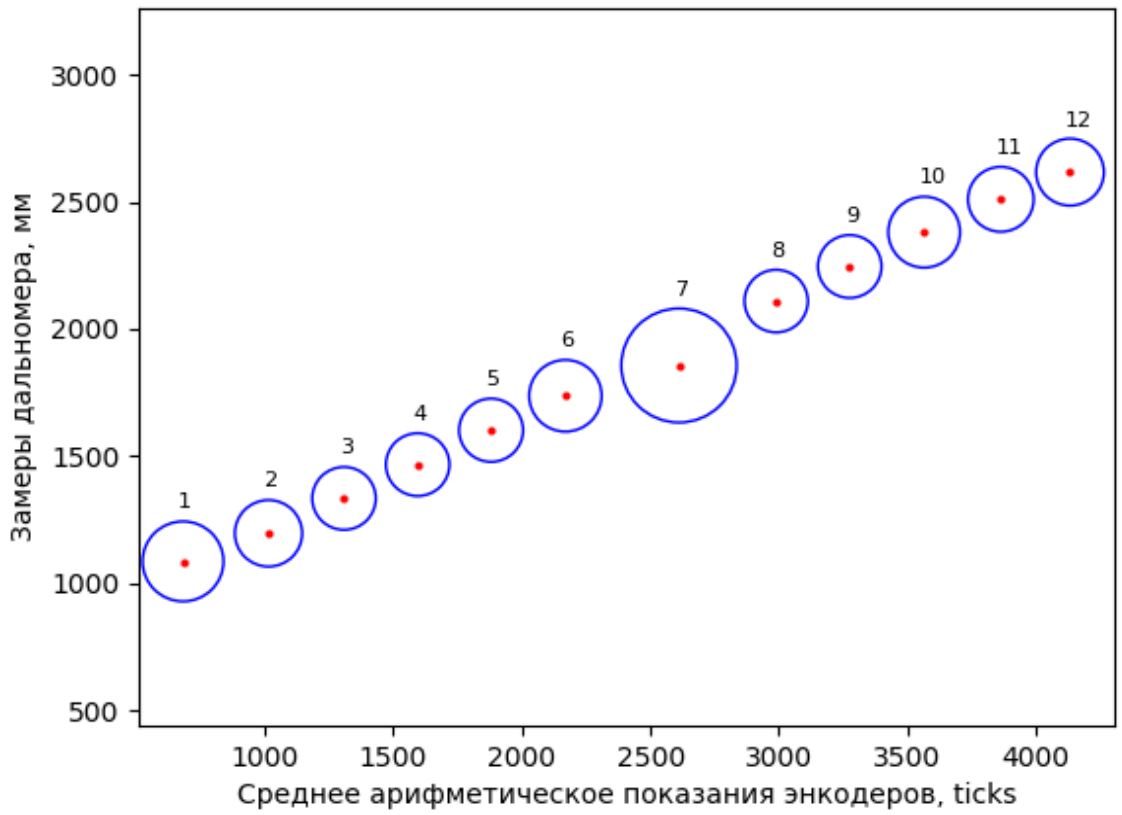

Рис. II.1.6: Цилиндры из набора данных N2

Пример программы-решения

Ниже представлено решение на языке Python 3

```
# -*- coding: utf-8 -*-
import math

def get_center(data):
  def get_kb(xy1, xy2, l):
    k = (xy1[0] - xy2[0]) / (xy2[1] - xy1[1])
    b = l[1] - k * l[0]
    return k, b

  def get_l(xy1, xy2):
    return (xy1[0] + xy2[0]) / 2, (xy1[1] + xy2[1]) / 2

  l1 = get_l(data[0], data[1])
  l2 = get_l(data[1], data[2])

  k1, b1 = get_kb(data[0], data[1], l1)
  k2, b2 = get_kb(data[1], data[2], l2)

  x = (b1 - b2) / (k2 - k1)
  y = k1 * x + b1

  return x, y
```

```
23
24  def calculate(lst):
25    dist_prev = 0
26    cylinders = [[]]
27    y = []
28    grow = False
29    for d in range(len(lst)):
30      encoder, distance = lst[d]
31
32      if distance == 3700:
33        if len(cylinders[-1]) == 2 and dist_prev != 3700:
34          cylinders[-1].append((encoder, y[-1]))
35          cylinders.append([])
36        dist_prev = distance
37        continue
38
39      dist_prev = distance
40      if len(y) > 0:
41        grow = distance <= y[-1]
42
43      if grow and len(cylinders[-1]) == 0:
44        cylinders[-1].append((encoder, y[-1]))
45
46      if not grow and len(cylinders[-1]) == 1:
47        cylinders[-1].append((encoder, y[-1]))
48
49      if grow and len(cylinders[-1]) == 2:
50        cylinders[-1].append((encoder, y[-1]))
51        cylinders.append([])
52
53      y.append(distance)
54
55    if len(cylinders[-1]) == 0:
56      cylinders.pop()
57
58    diams = []
59    for c in range(len(cylinders)):
60      center = get_center(cylinders[c])
61      radius = sorted([abs(cylinders[c][x][0] - center[0]) for x in range(3)])[1]
62      diams.append(radius * 2)
63
64    return diams.index(max(diams)) + 1
65
66  raw_data = []
67  for i in range(int(input())):
68    raw_data.append(list(map(float, input().split())))
69  print(calculate(raw_data))
```

Задача II.1.1.2. Цилиндры (5 баллов)

Робототехническое устройство, собранное по дифференциальной схеме и с заданным диаметром колес, движется по прямой с постоянной скоростью. На устройстве установлен ультразвуковой датчик расстояния, направленный влево перпендикулярно направлению движения.

Слева от робототехнического устройства установлены в ряд цилиндры разного диаметра d_i. Цилиндры расположены таким образом, что все их центры находятся на одной прямой. Высота цилиндров много выше высоты, на которой установлен

датчик расстояния. Цилиндры могут располагаться как вплотную друг к другу, так и с зазором.

Во время движения датчик расстояния, устроенный таким образом, что он имеет конус направленности $\alpha = 1^{''}$ и возвращает минимальное значение из заданного диапазона или максимально возможное в случае, если отсутствуют какие-либо объекты в непосредственной видимости датчика. Известно, что показания ультразвукового **датчика не идеальны**, в них всегда присутствует высокая доля помех. Частота опроса датчика — 10 Гц.

Необходимо найти цилиндр максимального диаметра. Известно, что радиус колес равен 0.09 м.

При движении робота не гарантируется, что он движется параллельно прямой, на которой установлены цилиндры. Гарантируется, что робот движется так, что во время движения фиксирует датчиком все установленные цилиндры таким образом, что было сделано не менее 3 изменений для каждого цилиндра.

Формат входных данных

Первая строка входных данных содержит одно целое число — N, где:

- N — количество измерений ($3 \leqslant N \leqslant 1000$).

Далее идут N строк, содержащие 2 вещественных числа через пробел — Enc, S, где:

- Enc — среднее арифметическое показаний энкодеров левого и правого колеса в градусах ($0 \leqslant Enc \leqslant 10^6$);
- S — показание датчика расстояния в мм ($10 \leqslant S \leqslant 3700$).

Формат выходных данных

Одна строка, содержащая одно целое число — номер цилиндра максимального диаметра считая по ходу движения.

Примеры

Примеры входных данных и ответов к ним можно найти по данной ссылке.

Решение

Решение задачи аналогично решению от предыдущей задачи "Цилиндры". Но с одним дополнением. Согласно условиям задачи, показания датчика расстояния **не идеальны**, т.е. имеют некий "разброс" значений. В реальных условиях это может быть вызвано несколькими факторами: влажностью, температурой, особенностями изготовления датчика и другими внешними и внутренними факторами.

Основная задача состоит в фильтрации значений.

Подробно тема фильтрации данных описана в материалах Университета Иннополис в разделе "Программирование интеллектуальных робототехнических систем" по адресу: Подраздел “Фильтрация значений датчиков”

Рассмотрим фильтрацию на примере набора данных N8. На рис. II.1.7 показаны исходные данные в виде графика.

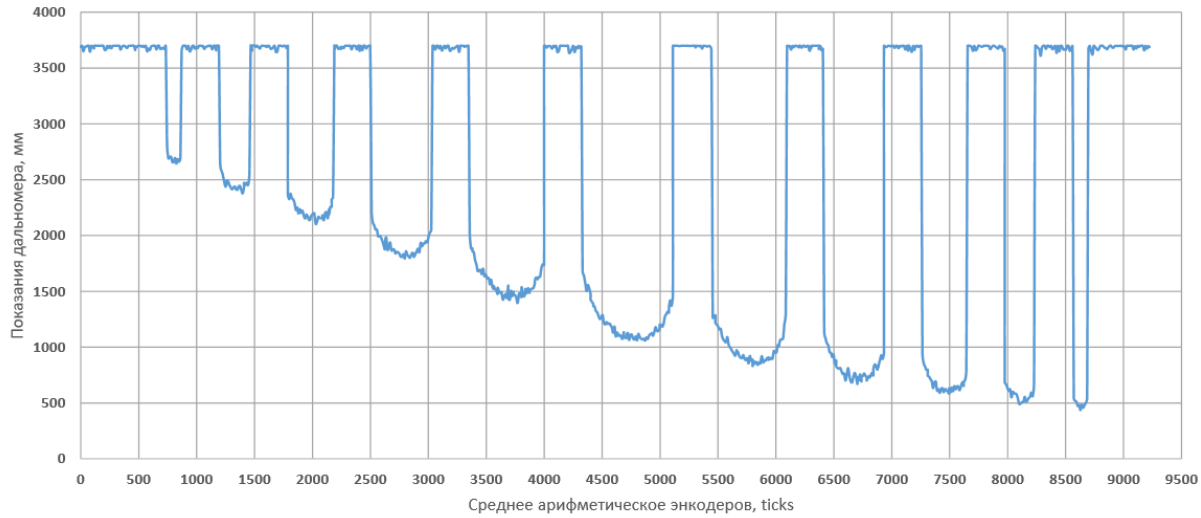

Рис. II.1.7: Данные из набора N8

Как заметно из графика, данные достаточно "шумные", что не позволяет однозначно определить начало и конец каждого цилиндра.

Пробуем провести фильтрацию данных. Сначала по алгоритму "Взвешенное скользящее среднее" с окном = 10, затем "простое скользящее среднее" с окном = 8. Но предварительно очистим "пустоты" между цилиндрами. Результат фильтрации показан на рис.

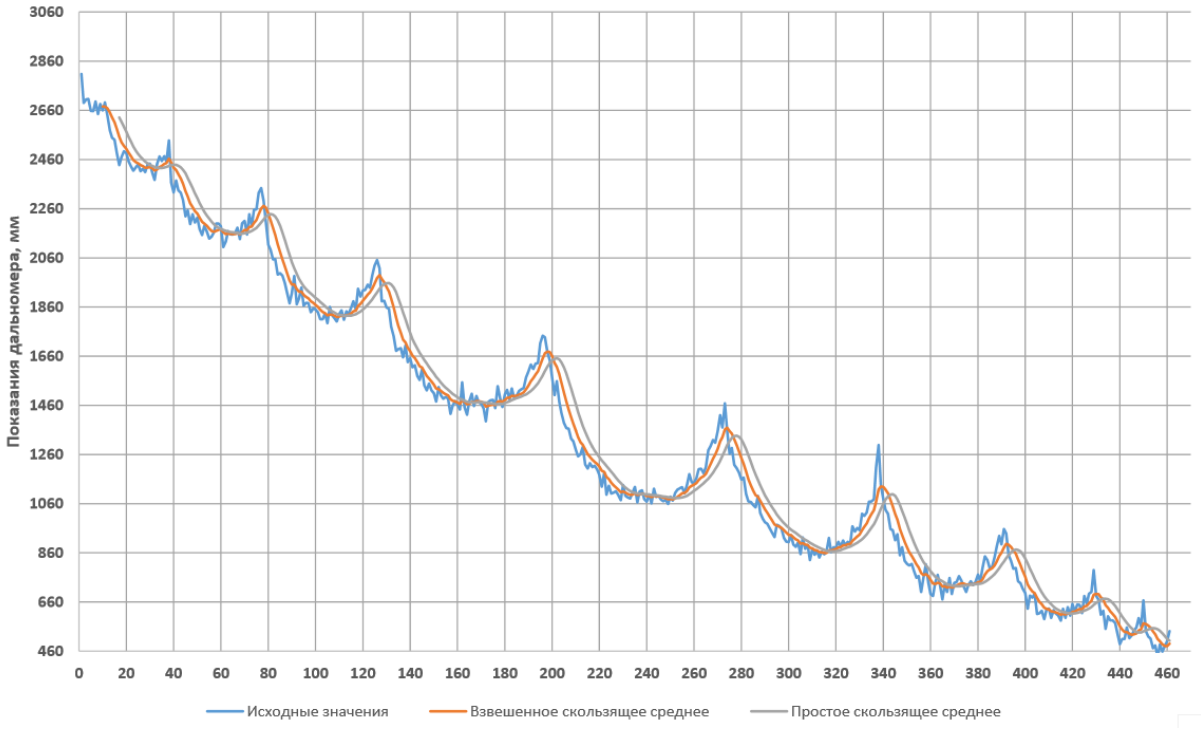

Рис. II.1.8: Данные из набора N8 после фильтрации

Теперь из полученных данных можно определить диаметры цилиндров по алгоритму из предыдущей задачи "Цилиндры".

Пример программы-решения

Ниже представлено решение на языке Python 3

```
# -*- coding: utf-8 -*-

def filters(data):
  def sma(lst, size=5):
    # Simple Moving Average
    return [sum(lst[i - size:i]) / size for i in range(size, len(lst))]

  def wma(lst, size=10):
    # Weighted moving average
    summ_w = sum(range(size + 1))
    res = []
    for x in range(len(lst) - size):
      dt = lst[x:size + x]
      res.append(sum([(i + 1) * dt[i] for i in range(len(dt))]) / summ_w)
    return res

  def median(lst, size=3):
    odd = size % 2 != 0
    res = []
    for i in range(size, len(lst)):
      l = sorted(lst[i - size:i])
      if odd:
        res.append(l[size // 2])
      else:
        res.append((l[size // 2] + l[size // 2 - 1]) / 2)
    return res

  def ema(lst, k_e=0.85):
    # Exponential Moving Average
    res = [lst[0]]
    for i in range(1, len(lst)):
      res.append(k_e * res[-1] + (1 - k_e) * lst[i])
    return res

  data = [i for i in data if 3600 > i > 10]
  return sma(wma(data, 10), 8)

def get_center(data):
  def get_kb(xy1, xy2, l):
    k = (xy1[0] - xy2[0]) / (xy2[1] - xy1[1])
    b = l[1] - k * l[0]
    return k, b

  def get_l(xy1, xy2):
    return (xy1[0] + xy2[0]) / 2, (xy1[1] + xy2[1]) / 2

  l1 = get_l(data[0], data[1])
  l2 = get_l(data[1], data[2])

  k1, b1 = get_kb(data[0], data[1], l1)
  k2, b2 = get_kb(data[1], data[2], l2)

  x = (b1 - b2) / (k2 - k1)
  y = k1 * x + b1

  return x, y
```

```
57
58  def calculate(lst_raw):
59    lst = filters(lst_raw)
60
61    cylinders, y = [[]], []
62    grow = False
63
64    for d in range(len(lst)):
65      distance = lst[d]
66
67      if len(y) > 0:
68        grow = distance <= y[-1]
69
70      if grow and len(cylinders[-1]) == 0:
71        cylinders[-1].append((d, y[-1]))
72
73      if not grow and len(cylinders[-1]) == 1:
74        cylinders[-1].append((d, y[-1]))
75
76      if grow and len(cylinders[-1]) == 2:
77        cylinders[-1].append((d, y[-1]))
78        cylinders.append([])
79
80      y.append(distance)
81
82    if len(cylinders[-1]) != 3:
83      cylinders.pop()
84
85    diams = []
86    for c in range(len(cylinders)):
87      center = get_center(cylinders[c])
88      radius = sorted([abs(cylinders[c][x][0] - center[0]) for x in range(3)])[1]
89      diams.append(radius * 2)
90
91    lengths = [
92        cylinders[i][1][0] - cylinders[i][0][0] for i in range(len(cylinders))
93    ]
94    return lengths.index(max(lengths)) + 1
95
96  data_raw = []
97  for i in range(int(input())):
98    data_raw.append(list(map(float, input().split()))[1])
99  print(calculate(data_raw))
```

Задача II.1.1.3. Цветные препятствия (10 баллов)

Даны два дифференциальных робота: Алиса и Боб. У Боба есть камера, а у Алисы — датчики препятствия. Данные роботы находятся в квадратном лабиринте с перегородками. Пол и ограждения — белые, а перегородки внутри лабиринта цветные. Перегородки или их линии образуют квадратные секции размером 250×250 мм, стороны которых параллельны стенкам.

Известно, что перегородки могут пересекаться и движения данных роботов ограничены следующими командами:

- F — проехать из центра одно сектора в центр следующего по ходу движения сектора;
- L — повернуться на месте налево, относительно текущего направления робота;

- R — повернуться на месте направо, относительно текущего направления робота.

Алиса начинает движение из неизвестной точки и останавливается где-то в лабиринте. Известны показания датчиков робота во время перемещения и его действия. Гарантируется, что по данным показаниям можно однозначно идентифицировать маршрут Алисы.

Перечислите последовательность в которой сменяются цвета препятствий перед роботом Бобом после каждой выполненной команды при его перемещении по оптимальному пути из точки старта в точку финиша. Гарантируется, что Боб сможет доехать до точки финиша. Путь считается оптимальным, если было сделано наименьшее количество действий(команд).

Характеристики роботов:

Камера Боба имеет угол обзора α и направленна по ходу движения робота. Высота и угол установки камеры таковы, что если бы перед роботом не было препятствий, то линия горизонта проходила бы горизонтально ровно по центру кадра. Высота перегородок в лабиринте значительно выше высоты установленной камеры.

Датчики препятствия, установленные на Алисе, расположены таким образом, что они позволяют определить есть ли слева, спереди или справа проезд в соседний сектор. Робот Боб и Алиса имеют вид цилиндров диаметром 250 мм однотонного чёрного цвета и занимает практически весь сектор. В случае если поле зрения датчиков Алисы попадет Боб, то он будет распознан как препятствие.

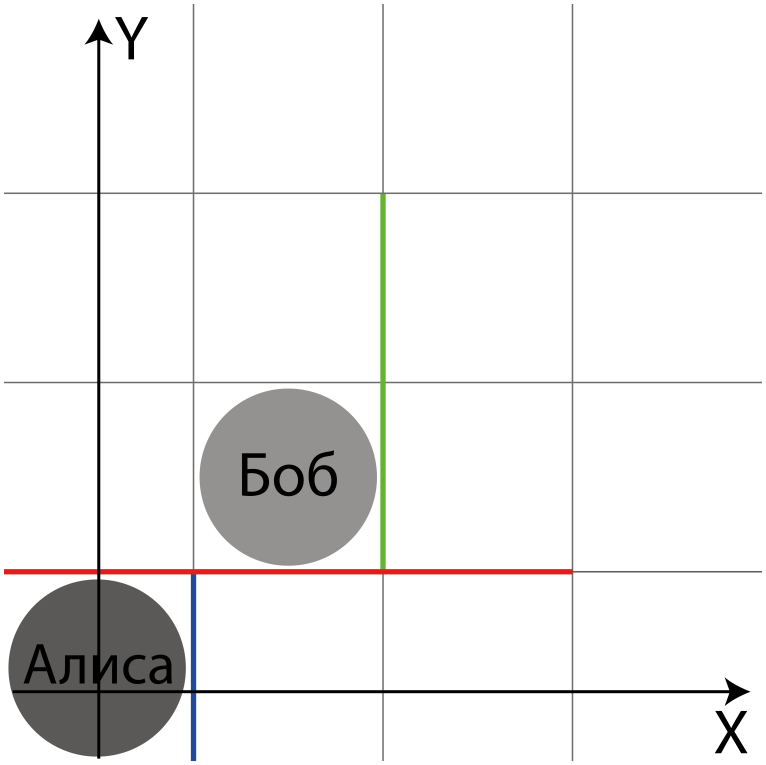

Рис. II.1.9: Пример начального расположения Боба и Алисы

Формат входных данных

Первая строчка содержит четыре целых числа через пробел – H, N, K, α где:

- H — сторона поля в мм ($10^3 \leqslant H \leqslant 10^6$);
- N — количество цветных перегородок ($1 \leqslant N \leqslant 100$);
- K — количество движений Алисы ($1 \leqslant K \leqslant 30$);
- α — угол обзора камеры ($5° \leqslant \alpha \leqslant 25°$).

Вторая строчка содержит одну букву и 4 целых числа через пробел — D, X_s, Y_s, X_e, Y_e, где:

- D — начальная ориентация Боба ($D \in \{U, D, L, R\}$, где U - по направлению оси Y, D - против направления оси Y, L - против направления оси X, R - по направлению оси X)
- X_s — начальная координата X_s Боба в мм ($0 \leqslant X_s \leqslant H$, $X_s = 125+250a$, $a \in \mathbb{N}$);
- Y_s — начальная координата Y_s Боба в мм ($0 \leqslant Y_s \leqslant H$, $Y_s = 125 + 250a$, $a \in \mathbb{N}$);
- X_e — конечная координата X_e Боба в мм ($0 \leqslant X_e \leqslant H$, $X_e = 125+250a$, $a \in \mathbb{N}$);
- Y_e — конечная координата Y_e Боба в мм ($0 \leqslant X_e \leqslant H$, $Y_e = 125 + 250a$, $a \in \mathbb{N}$);

Далее идут N строк, содержащие 4 целых числа и код цвета через пробел — $X_{i,1}$, $Y_{i,1}$, $X_{i,2}$, $Y_{i,2}$, Col_i, где:

- $X_{i,1}$ — координата $X_{i,1}$ начальной точки перегородки в мм ($0 \leqslant X_{i,1} \leqslant H$, $X_{i,1} = 250a$, $a \in \mathbb{N}$);
- $Y_{i,1}$ — координата $Y_{i,1}$ начальной точки перегородки в мм ($0 \leqslant Y_{i,1} \leqslant H$, $Y_{i,1} = 250a$, $a \in \mathbb{N}$);
- $X_{i,2}$ — координата $X_{i,2}$ конечной точки перегородки в мм ($0 \leqslant X_{i,2} \leqslant H$, $X_{i,2} = 250a$, $a \in \mathbb{N}$);
- $Y_{i,2}$ — координата $Y_{i,2}$ конечной точки перегородки в мм ($0 \leqslant Y_{i,2} \leqslant H$, $Y_{i,2} = 250a$, $a \in \mathbb{N}$);
- Col_i — код цвета данной перегородки в шестнадцатеричной системе счисления ($000000 \leqslant Col_i < FFFFFF$);

Далее идут K строк, содержащие 3 целых числа и одну букву через пробел — S_l, S_f, S_r, Act, где:

- S_l — показания датчика обнаружения препятствия, направленного влево ($S_l \in \{0, 1\}$, где 0 — препятствие не обнаружено);
- S_f — показания датчика обнаружения препятствия, направленного вперёд ($S_f \in \{0, 1\}$, где 0 — препятствие не обнаружено);
- S_r — показания датчика обнаружения препятствия, направленного вправо ($S_r \in \{0, 1\}$, где 0 — препятствие не обнаружено);
- Act — команда выполненная роботом после получения данных с датчиков ($Act \in \{F, L, R\}$).

Формат выходных данных

Одна строка, содержащая последовательность кодов цветов (исключая белый), которые будут менять друг друга в камере Боба после завершения каждого из действия, выполненного во время его перемещения. Разделитель между кодами цветов — пробел.

Примеры

Пример №1

Стандартный ввод
2000 3 11 15 U 875 875 125 375 250 1250 250 250 FF0000 250 250 1000 250 0000FF 1500 500 1500 1250 FFFF00 0 0 1 F 0 0 1 F 0 0 1 F 0 0 0 F 0 1 0 R 1 0 0 F 1 0 0 F 1 0 0 F 1 0 1 F 1 0 1 F 1 0 1 F
Стандартный вывод
000000

Пример №2

Стандартный ввод
1500 4 7 15 D 125 875 375 125 0 750 500 750 FF0000 250 0 250 250 FFFF00 250 250 1000 250 0000FF 750 500 750 750 00FF00 1 0 0 R 0 0 0 F 0 0 1 F 0 1 1 L 0 0 1 F 0 0 1 F 1 1 1 L
Стандартный вывод
FF0000 FFFF00

Комментарии

Оптимальным считается маршрут, длина которого, состоящая из возможных команд, минимальна.

Решение

Декомпозиция решения задачи:

1. Составить карту поля с учетом цветов перегородок
2. Локализоваться роботом Алиса (определить конечный сектор после выполнения указанных команд перемещения)
3. Определить оптимальный маршрут перемещения робота Боб в заданный конечный сектор
4. Во время перемещения робота Боба в конечный сектор отслеживать цвета перегородок, встречающие на пути

В данной задаче поле представлено в виде лабиринта. Координаты установки перегородок и их цвета определяются согласно условиям задачи.

Для представления карты местности, в т.ч. и для лабиринта, можно использовать графы, точнее матрицу смежности. Более подробно про варианты представления местности в цифровом виде описано в материалах Университета Иннополис в разделе "Программирование интеллектуальных робототехнических систем" по адресу: Подраздел “Построение карты. Локализация.”

После составления карты начинаем локализоваться роботом Алиса. По условиям задачи начальные координаты секции старта Алисы не заданы, но указаны состояния датчиков расстояния (с трех сторон) и команды перемещения.

Как вариант, можно "вставать" в каждую секцию лабиринта в каждом из четырех направлений (север, восток, юг, запад) и сравнивать состояние датчиков из задания с матрицей смежности лабиринта, если они совпадают, то “выполняем” указанное действие и переходим к сравнению следующих состояний. Если не совпадают - меняем направление и/или секцию. В задании точно есть одна секция в определенном направлении, где все совпадает. После того, как определим секцию старта робота Алисы, мы должны еще вычислить ее конечную секцию, т.к. робот Алиса для робота Боб является препятствием черного цвета. Для этого мы должны учитывать смену координат и направления робота при выполнении указанных действий для Алисы.

После локализации Алисы второй робот - Боб вычисляет оптимальный маршрут перемещения от секции старта до конечной секции. Более подробно про расчеты оптимального пути написано в материалах Университета Иннополис в разделе "Программирование интеллектуальных робототехнических систем" по адресу: Подраздел “Планирование и построение маршрута”

После составления пути, начинаем по нему перемещаться роботом “Боб” и отслеживать цвета стенок, которые встречаются перед ним.

В качестве результата нужно вывести цвета стенок, которые видел робот Боб во время передвижения.

Рассмотрим пример на наборе данных N8.

На рис. II.1.10 показан лабиринт с расставленными стенками и их цветами, на-

чальный сектор старта робота Боб и начальный сектор робота Алиса (уже после локализации)

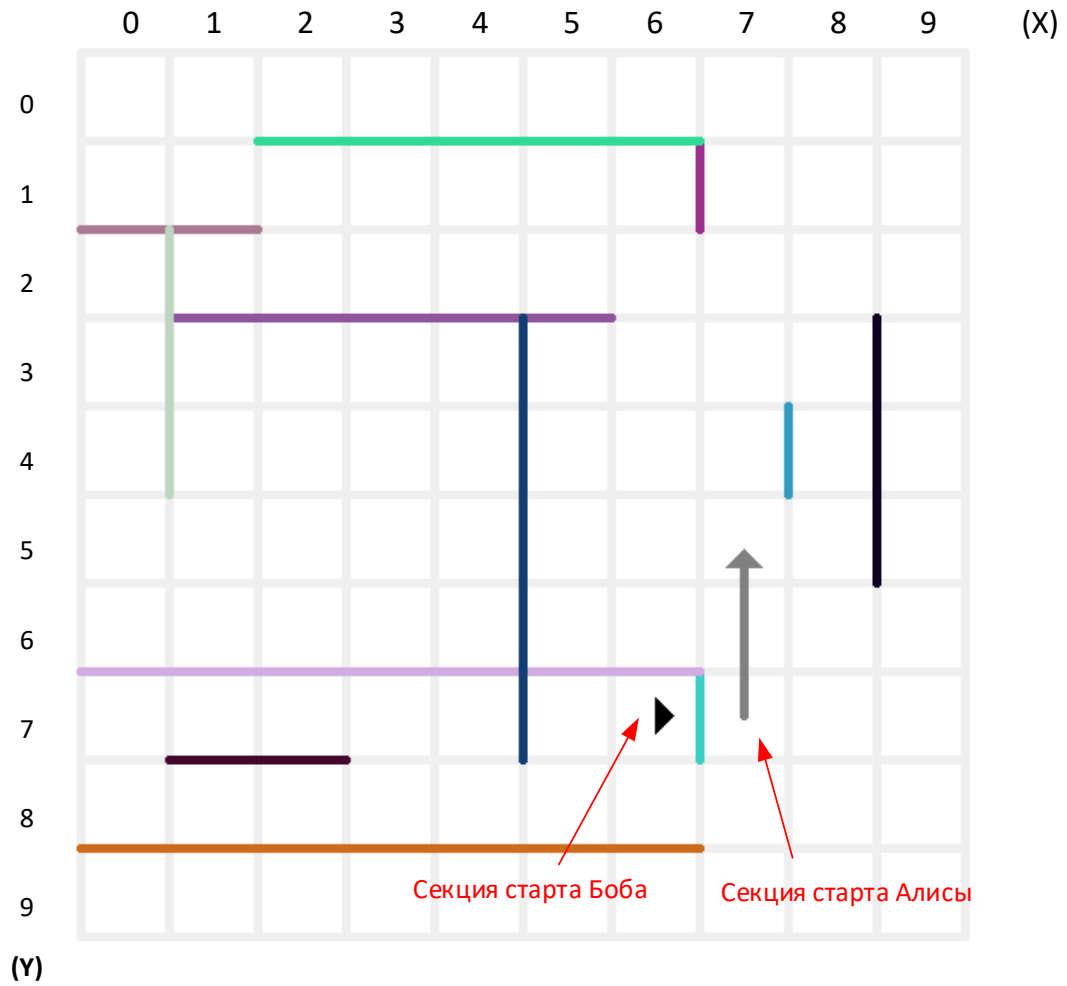

Рис. II.1.10: Расположение перегородок в лабиринте и положение роботов

На рис. II.1.11 показаны пути перемещения роботов Алиса и Боб и пунктирными кругами показаны перегородки, встречающие на пути робота Боб и их порядок.

Ответ в данной задаче на наборе данных N8: $3ACFC4\ \ CB6B1B$ 000000 $BDD5C1$

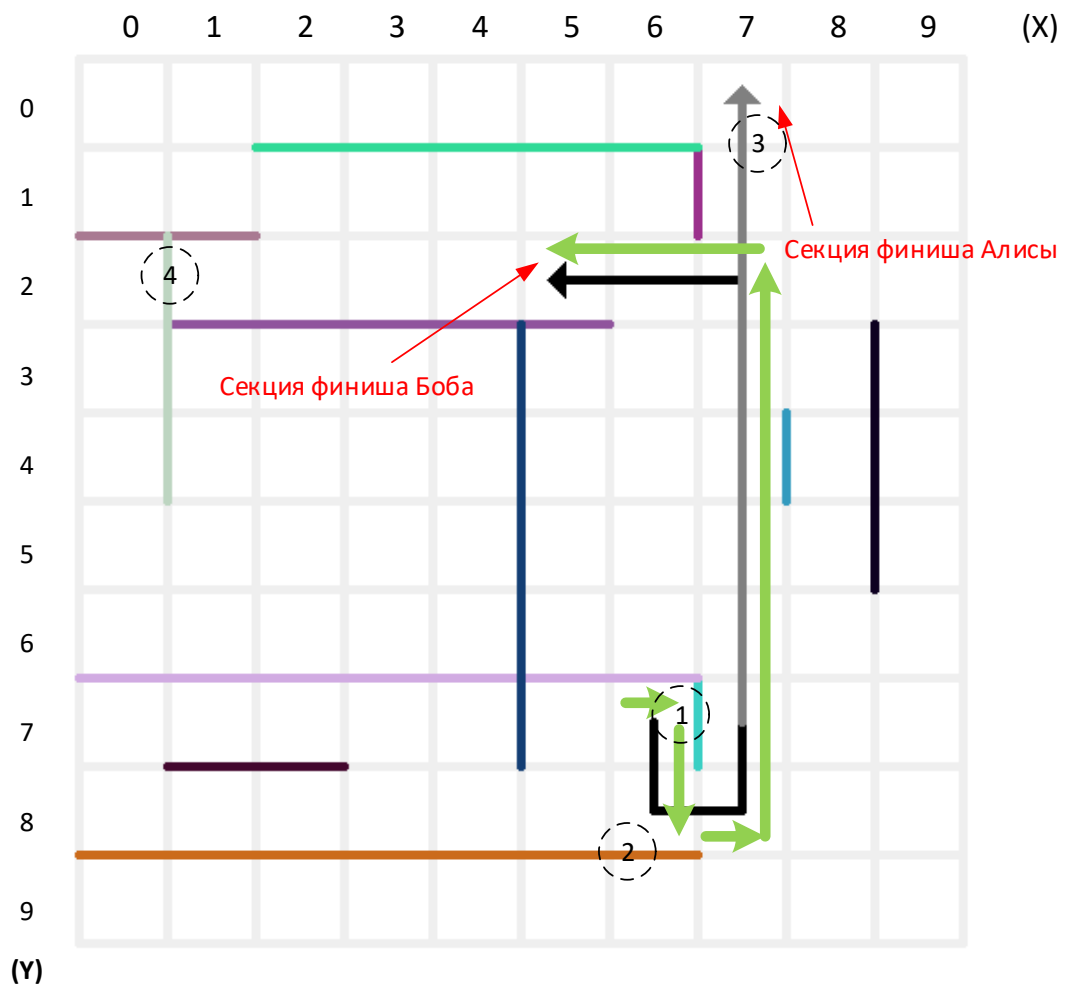

Рис. II.1.11: Положения роботов после выполнения заданий

Пример программы-решения

Ниже представлено решение на языке Python 3

```
# -*- coding: utf-8 -*-

import sys


def set_walls(datas):
  # расставляем перегородки
  global MAP

  for _ in range(len(datas)):
    Xs, Ys, Xe, Ye, clr = datas.pop(0).split()
    Xs, Ys, Xe, Ye = map(lambda x: int(x) // cell_size, (Xs, Ys, Xe, Ye))

    min_y, max_y = min(Ys, Ye), max(Ys, Ye)
    min_x, max_x = min(Xs, Xe), max(Xs, Xe)

    if Xs == Xe:  # вертикальная стена
      for y in range(min_y, max_y):
        cell1 = cells_by_task[xy_to_cell(y, Xs)]
```

```
20          cell2 = cells_by_task[xy_to_cell(y, Xs - 1)]
21          map_update(cell1, cell2, clr)
22
23      if Ys == Ye:  # горизонтальная стена
24        for x in range(min_x, max_x):
25          cell1 = cells_by_task[xy_to_cell(Ys, x)]
26          cell2 = cells_by_task[xy_to_cell(Ys - 1, x)]
27          map_update(cell1, cell2, clr)
28
29
30  def xy_to_cell(yy, xx):
31    return yy * map_size + xx
32
33
34  def cell_to_xy(cell):
35    yy = cell // map_size
36    xx = cell - yy * map_size
37    return yy, xx
38
39
40  def astar(start, end):
41    def hh(cur):
42      # расчет расстояния от текущей секции до конечной
43      xy_cur = cell_to_xy(cur)
44      xy_end = cell_to_xy(end)
45      return (abs(xy_cur[0] - xy_end[0]) + abs(xy_cur[1] - xy_end[1])) * 10
46
47    def gg(cur):
48      xy_start = cell_to_xy(start)
49      xy_cur = cell_to_xy(cur)
50      return (abs(xy_cur[0] - xy_start[0]) + abs(xy_cur[1] - xy_start[1])) * 10
51
52    visited = [False] * map_len
53    G = [float('inf')] * map_len  # расст от начала до текущей
54    H = [10**3] * map_len  # расст от текущей до начала
55    F = [10**3] * map_len
56
57    G[start] = 0
58    H[start] = hh(start)
59    F[start] = G[start] + H[start]
60
61    lst = [start]
62    pairs = []
63
64    done = False
65    step = 0
66    while len(lst):
67      minn = 10**3
68      # выбираем вершину из списка lst с меньшим весом
69
70      if step == 0:
71        cells = cells_around[start]
72        cell = cells[bob_dir]
73        if cell in lst:
74          for c in range(len(cells)):
75            if cells[c] in lst:
76              #print (c, bob_dir,cells[c], (bob_dir - c) % 3 )
77              dop = (bob_dir - c) % 4
78              dop == 1 if dop == 3 else dop
79              #G[cells[c]] = dop
```

```
80              #F[cells[c]] = dop
81              F[cells[c]] = G[cells[c]] + H[cells[c]]  #* 0.2 + dop * 0.8
82          G[lst[lst.index(cell)]] = 0
83          F[lst[lst.index(cell)]] = 0
84
85      for v in range(len(lst)):
86        if F[lst[v]] < minn:
87          current = lst[v]
88          minn = F[lst[v]]
89
90      for p in range(map_len):
91        if MAP[current][p] is True and not visited[p]:
92          if G[p] > G[current] + 10:  #MAP[current][p]:
93            G[p] = G[current] + 10  #MAP[current][p]
94            H[p] = hh(p)
95            F[p] = G[p] + H[p]
96            pairs.append([p, current])
97            if p == end:
98              #print ('end:',p, end)
99              done = True
100             break
101
102         if p not in lst:
103           lst.append(p)
104     step += 1
105     if done:
106       break
107
108     visited[current] = True
109     lst.remove(current)
110
111   if len(pairs) < 2:
112     return []
113
114   prev = pairs[-1][1]
115   path_ = [pairs[-1][0], prev]
116
117   for p in range(len(pairs)):
118     for w in range(len(pairs)):
119       if pairs[w][0] == prev:
120         prev = pairs[w][1]
121         path_.append(prev)
122         break
123
124   return path_[::-1]
125
126
127 def bfs(start, end):
128   visited = [False for i in range(map_len)]
129
130   path = []
131   queue = [start]
132   step = 0
133   while len(queue) > 0:
134     p = queue.pop(0)
135
136     if step == 0:
137       cells = cells_around[p]
138       cell = cells[bob_dir]
139       if cell in lst:
```

```
        for c in range(len(cells)):
          if cells[c] in lst:
            dop = (bob_dir - c) % 4
            dop == 1 if dop == 3 else dop
            F[cells[c]] = G[cells[c]] + H[cells[c]] * 0.2 + dop * 0.8

    if not visited[p]:
      visited[p] = True
      path.append(p)

      for i in range(len(MAP)):
        if not visited[i] and MAP[p][i] is True:
          queue.append(i)
    if (p == end):
      break
  path.reverse()
  back = [path[0]]
  for p in path[1:]:
    if MAP[back[-1]][p] is True:
      back.append(p)
  back.reverse()
  return back


def map_update(cell1, cell2, clr):
  if cell1 is None or cell2 is None:
    return

  global MAP
  if MAP[cell1][cell2] in ('FFFFFF', True, False, '000000'):
    MAP[cell1][cell2] = clr
    MAP[cell2][cell1] = clr


def cell_update_walls(cell1, clr):
  cells = cells_around[cell1]
  for cell2 in cells:
    if cell2 is not None and cell2 != alice_cell_end:
      map_update(cell1, cell2, clr)


def check_add_clr(cell1, dir):
  global looked
  cell2 = cell1
  while True:
    cell2 = cells_around[cell2][dir]

    if cell2 == cell1 or cell2 is None:
      break

    if MAP[cell1][cell2] not in (True, False, 'FFFFFF'):
      looked.append(MAP[cell1][cell2])
      if len(looked) > 1 and looked[-1] == looked[-2]:
        looked.pop()
      break

    cell1 = cell2


def solution(data):
```

```
200     global map_size, map_len, X0, Y0, MAP, walls, looked, bob_dir, \
201         alice_states, alice_moves, alice_cell_end, cells_by_task, cells_around, heads
202
203     looked = []
204
205     # parsing data START
206     H, N, K, alpha = map(int, data.pop(0).split())
207
208     map_size = H // cell_size
209     map_len = map_size * map_size
210
211     MAP = [[False for _ in range(map_len)] for _ in range(map_len)]
212
213     # нумерация секторов "как в задаче"
214     cells_by_task = sum(
215         sorted([[y * map_size + x for x in range(map_size)]
216                 for y in range(map_size)],
217                reverse=True), [])
218
219     BOBs = data.pop(0).strip().split()
220     bob_dir = 'URDL'.find(BOBs.pop(0))
221     bob_Xs, bob_Ys, bob_Xe, bob_Ye = map(lambda n: int(n) // 250, BOBs)
222     alice_cell_end = -1
223
224     # перегородки
225     set_walls(data[:N])
226
227     # Alice data
228     alice_states, alice_moves = [], []
229     for i in range(K):
230       states = data[i + N].strip().split()
231       alice_states.append(list(map(int, states[:3])))
232       alice_moves.append(states[3])
233
234     cells_around = []  # cells neighbours
235     for cell in range(map_len):
236       yy, xx = cell_to_xy(cell)
237       cells = [None] * 4
238       if yy - 1 >= 0: cells[0] = xy_to_cell(yy - 1, xx)
239       if xx + 1 < map_size: cells[1] = xy_to_cell(yy, xx + 1)
240       if yy + 1 < map_size: cells[2] = xy_to_cell(yy + 1, xx)
241       if xx - 1 >= 0: cells[3] = xy_to_cell(yy, xx - 1)
242       cells_around.append(cells)
243
244       for cell2 in cells:
245         map_update(cell, cell2, True)
246
247     # parsing data END
248
249     # start
250     bob_cell_start = cells_by_task[xy_to_cell(bob_Ys, bob_Xs)]
251     bob_cell_end = cells_by_task[xy_to_cell(bob_Ye, bob_Xe)]
252     #print('BOB start, end, dir:',bob_cell_start, bob_cell_end, bob_dir)
253
254     # сектор старта Боба (открытые стенки -> черный цвет)
255     cell_update_walls(bob_cell_start, '000000')
256
257     # Alice localize
258     sensors_global = []
259     for cell in range(map_len):
```

```
260       s = [1, 1, 1, 1]  # 1 - wall, 0 - free
261       cells = cells_around[cell]
262
263       for i in range(4):
264         if cells[i] is not None and MAP[cell][cells[i]] is True:
265           s[i] = 0
266       s = [s[3]] + s[:3]
267
268       sensors_global.append([])
269       for _ in range(4):
270         sensors_global[-1].append(s[:3])
271         s.append(s.pop(0))
272
273     hypoteses = []
274     for cell in range(len(sensors_global)):
275       if alice_states[0] in sensors_global[cell] and cell != bob_cell_start:
276         for d in range(4):
277           if alice_states[0] == sensors_global[cell][d]:
278             hypoteses.append((cell, d, alice_states[0]))
279     #print (hypoteses)
280     alice_states.append([])
281
282     for h in hypoteses:
283       cell0, dir0, state0 = h
284       cell, dir = cell0, dir0
285       fired = False
286       moves = [-map_size, 1, map_size, -1]
287
288       for m in range(len(alice_moves)):
289         move = alice_moves[m]
290         if move == 'R':
291           dir = (dir + 1) % 4
292         elif move == 'L':
293           dir = (dir + 3) % 4
294         elif move == 'F':
295           cell += moves[dir]
296
297         if sensors_global[cell][dir] != alice_states[m + 1]:
298           fired = True
299           break
300
301       if not fired or m == len(alice_moves) - 1:
302         break
303
304     #print('Alice start -> end:',cell0, dir0, state0, '->', cell, dir,
       ↪  sensors_global[cell][dir])
305     alice_cell_end = cell
306
307     cell_update_walls(cell, '000000')  # "walls" around Alice's cell
308     cell_update_walls(bob_cell_start, True)  # remove Bob 'black' walls
309
310     bob_path = astar(bob_cell_start, bob_cell_end)
311     bob_cell = bob_cell_start
312
313     for step in bob_path[1:]:
314       way = step - bob_cell
315       way_dir = [-map_size, 1, map_size, -1].index(way)
316
317       if way_dir != bob_dir:
318         b_w = str(bob_dir) + str(way_dir)
```

```
319       turn_cw = False if b_w in ('03', '10', '21', '32') else True
320
321       for _ in range(3):
322         check_add_clr(bob_cell, bob_dir)
323         if turn_cw:
324           bob_dir = (bob_dir + 1) % 4
325         else:
326           bob_dir = (bob_dir + 3) % 4
327
328         if way_dir == bob_dir: break
329
330     check_add_clr(bob_cell, bob_dir)
331     bob_cell = step
332
333   return looked
334
335
336 # not work: 1, 6, 7?, 11?
337
338 data = sys.stdin.readlines()
339 print(solution(data))
```

Задача II.1.1.4. Цветные препятствия 2 (10 баллов)

Дан дифференциальный робот: Боб. У Боба есть камера, имеющая угол обзора α, направленная по ходу движения робота и находящаяся в центре робота на его оси вращения. Высота и угол установки камеры таковы, что если бы перед роботом не было препятствий, то линия горизонта проходила бы горизонтально ровно по центру кадра. Высота перегородок в лабиринте значительно выше высоты установленной камеры.

Боб находится в квадратном лабиринте с перегородками. Пол и ограждения — белые, а перегородки внутри лабиринта цветные. Перегородки или их линии образуют квадратные секции размером 250×250 мм, стороны которых параллельны стенкам.

Перечислите последовательность в которой сменяются цвета препятствий попавших в камеру робота при его вращении на угол β, относительно начального положения. Известно, что препятствие считается попавшим если оно присутствует на 0.1 части изображения. В случае, если в камере присутствует несколько перегородок, то их цвета следует выводить в направлении вращения робота.

Формат входных данных

Первая строчка содержит два целых числа и одно вещественное число через пробел – H, N, α, где:

- H — сторона поля в мм ($10^3 \leqslant H \leqslant 10^6$);
- N — количество цветных перегородок ($1 \leqslant N \leqslant 100$);
- α — угол обзора камеры в рад ($0.1 \leqslant \alpha \leqslant 0.5$).

Вторая строчка содержит одну букву, два целых числа и одно вещественное число через пробел — D, X_s, Y_s, β, где:

- D — начальная ориентация Боба ($D \in \{U, D, L, R\}$, где U - по направлению оси Y, D - против направления оси Y, L - против направления оси X, R - по

направлению оси X)

- X_s — начальная координата X_s Боба в мм ($0 \leqslant X_s \leqslant H$, $X_s = 125+250a$, $a \in \mathbb{N}$);
- Y_s — начальная координата Y_s Боба в мм ($0 \leqslant Y_s \leqslant H$, $Y_s = 125 + 250a$, $a \in \mathbb{N}$);
- β — угол на который необходимо повернуться Бобу, в рад ($-2\pi \leqslant \beta \leqslant 2\pi$, при $\beta > 0$ – вращение происходит против часовой стрелки).

Далее идут N строк, содержащие 4 целых числа и код цвета через пробел — $X_{i,1}$, $Y_{i,1}$, $X_{i,2}$, $Y_{i,2}$, Col_i, где:

- $X_{i,1}$ — координата $X_{i,1}$ начальной точки перегородки в мм ($0 \leqslant X_{i,1} \leqslant H$, $X_{i,1} = 250a$, $a \in \mathbb{N}$);
- $Y_{i,1}$ — координата $Y_{i,1}$ начальной точки перегородки в мм ($0 \leqslant Y_{i,1} \leqslant H$, $Y_{i,1} = 250a$, $a \in \mathbb{N}$);
- $X_{i,2}$ — координата $X_{i,2}$ конечной точки перегородки в мм ($0 \leqslant X_{i,2} \leqslant H$, $X_{i,2} = 250a$, $a \in \mathbb{N}$);
- $Y_{i,2}$ — координата $Y_{i,2}$ конечной точки перегородки в мм ($0 \leqslant Y_{i,2} \leqslant H$, $Y_{i,2} = 250a$, $a \in \mathbb{N}$);
- Col_i — код цвета данной перегородки в шестнадцатеричной системе счисления ($000000 \leqslant Col_i < FFFFFF$);

Формат выходных данных

Одна строка, содержащая последовательность кодов цветов (исключая белый), которые будут менять друг друга в камере Боба во время его вращения, через пробел.

Примеры

Примеры входных данных и ответов к ним можно найти по данной ссылке.

Решение

Начнем решение задачи с формирования модели лабиринта, которая будет включать в себя координаты перегородок, стенок и их цвета (по условиям задачи все стены и пол - белого цвета). По сути, это отрезки с известными координатами.

Нам также известны:

1. Координаты установки робота и его направление
2. Угол, на который должен повернуться робот и направление вращения
3. Угол обзора камеры и

Для примера возьмем набор данных N1. Робот должен повернуться против часовой стрелки на 1.22 радиана (почти 70°). Угол обзора камеры - 0.01 радиана. С некоторой дискретностью, например, в 3° начинаем “вращаться” - проверять пересечение отрезков (“луча камеры” робота и стенок лабиринта). Не учитываем повторяющие друг за другом цвета и белый цвет. См. рис. II.1.12.

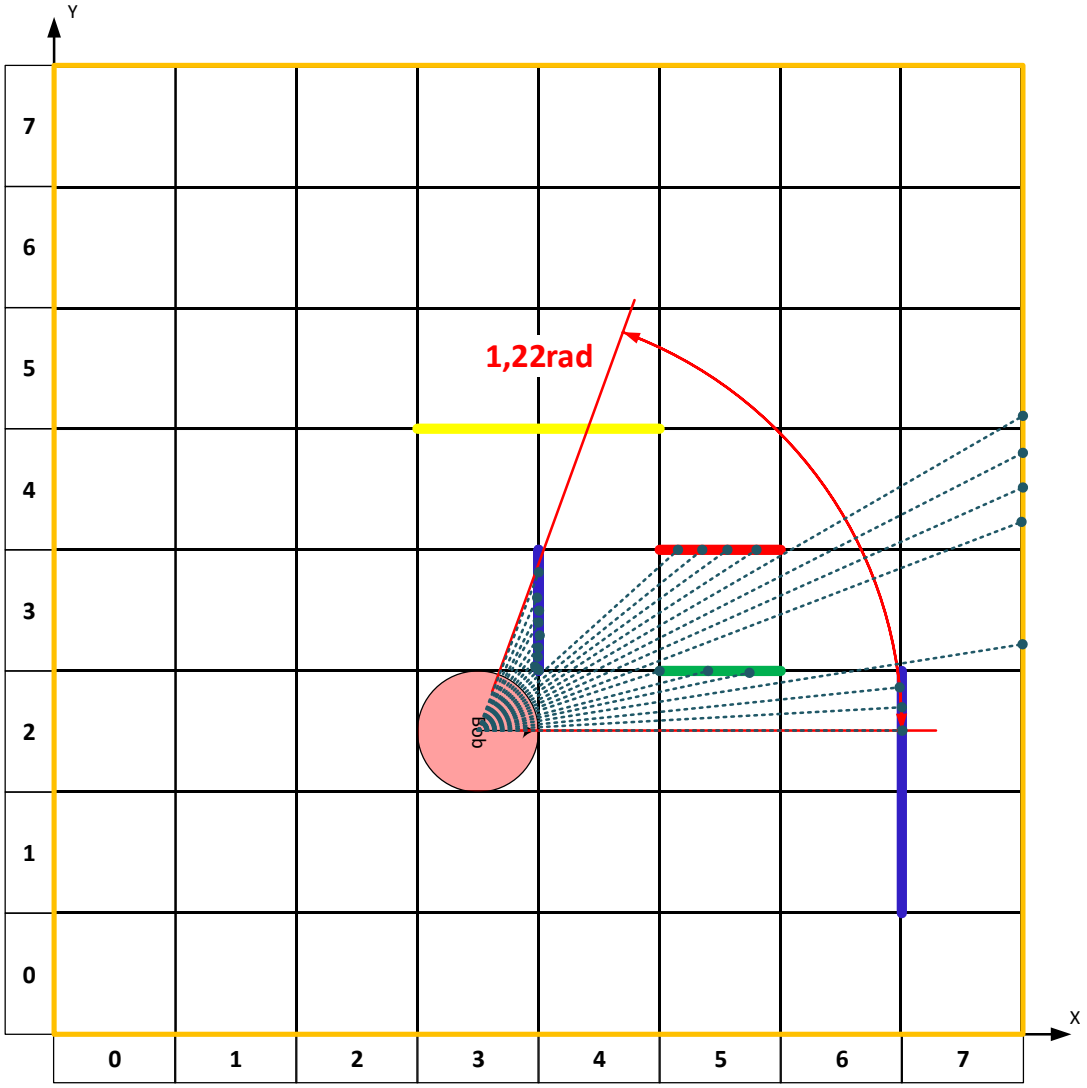

Рис. II.1.12: Пример сканирования стенок при повороте робота против часовой стрелки

В качестве ответа выводим список “замеченных” камерой цветов через пробел. В нашем примере ответ получается: $0000FF$ $00FF00$ $FF0000$ $0000FF$ (синий, зеленый, красный, синий).

С более широким углом камеры проверяем пересечение луча с ближайшими стенками внутри зоны видимости камеры, при этом не забываем “поворачивать” камеру в заданном направлении до заданного угла (рис. II.1.13)

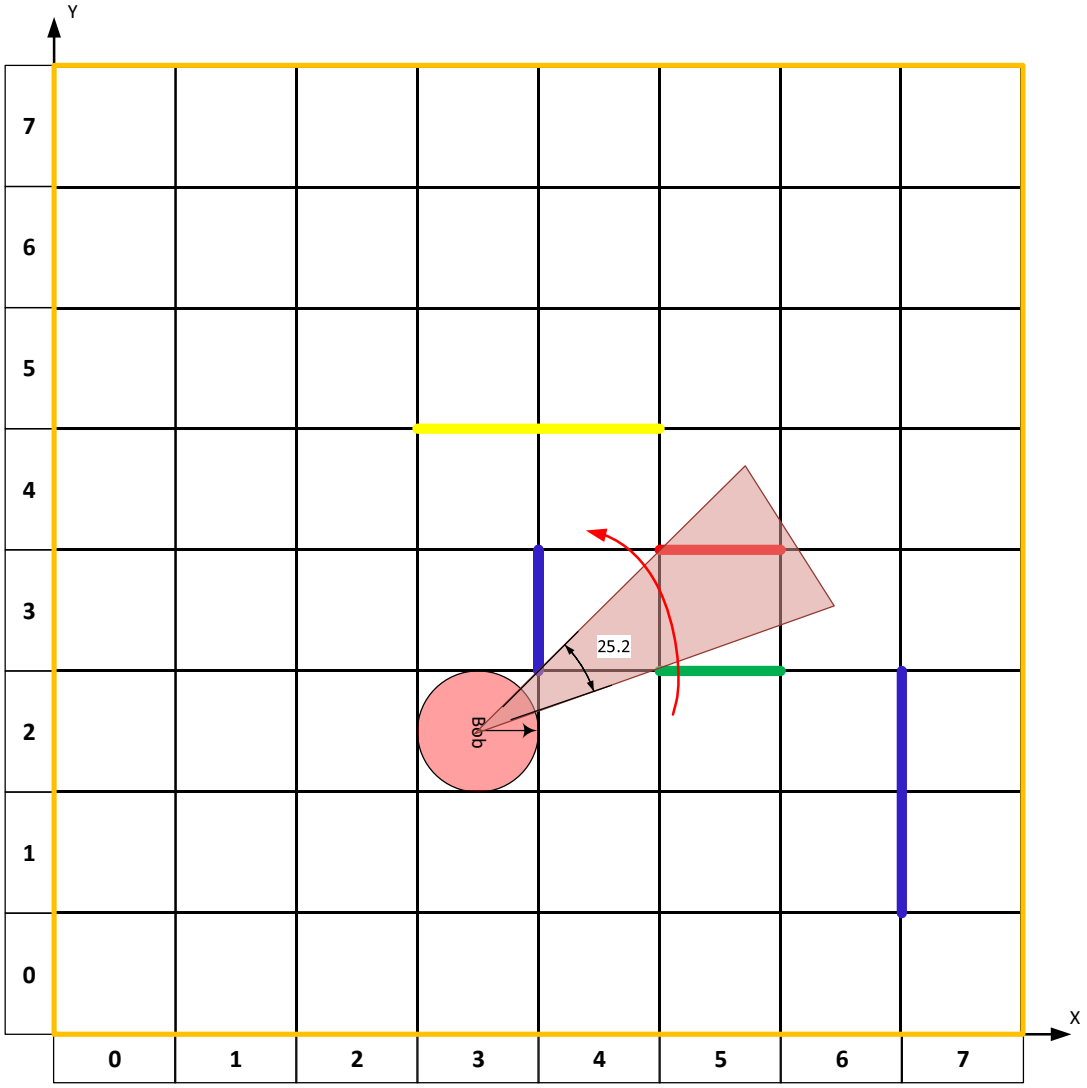

Рис. II.1.13: Пример зоны видимости камеры с углом обзора 25.2° (0.43 радиан)

Ниже представлено решение на языке Python 3

```
import numpy as np
from enum import IntEnum
import math


class Dir(IntEnum):
  LEFT = 0
  UP = 1
  RIGHT = 2
  DOWN = 3


def det(a, b):
  return a[0] * b[1] - a[1] * b[0]


```

```
17  def line_intersection(line1, line2):
18    xdiff = (line1[0][0] - line1[1][0], line2[0][0] - line2[1][0])
19    ydiff = (line1[0][1] - line1[1][1], line2[0][1] - line2[1][1])
20    div = det(xdiff, ydiff)
21    if div == 0:
22      return False
23    ua = ((line2[1][0] - line2[0][0]) * (line1[0][1] - line2[0][1]) -
24          (line2[1][1] - line2[0][1]) * (line1[0][0] - line2[0][0])) / div
25    ub = ((line1[1][0] - line1[0][0]) * (line1[0][1] - line2[0][1]) -
26          (line1[1][1] - line1[0][1]) * (line1[0][0] - line2[0][0])) / div
27
28    if ua < 0 or ua > 1 or ub < 0 or ub > 1:
29      return False
30
31    d = (det(*line1), det(*line2))
32    x = det(d, xdiff) / div
33    y = det(d, ydiff) / div
34    return x, y
35
36
37  def raycast(point1, point2, lines, skip=False):
38    line1 = (point1, point2)
39    intersections = []
40    for curline in range(len(lines)):
41      point3 = (lines[curline][0], lines[curline][1])
42      point4 = (lines[curline][2], lines[curline][3])
43      line2 = (point3, point4)
44      intersectresult = line_intersection(line1, line2)
45      if intersectresult:
46        intersections.append((intersectresult, lines[curline][4], curline))
47    if len(intersections) > 1:
48      intersections.sort(key=lambda x: np.sqrt((point1[0] - x[0][0]) * (point1[
49          0] - x[0][0]) + (point1[1] - x[0][1]) * (point1[1] - x[0][1])))
50    if skip:
51      if len(intersections) > 1:
52        return intersections[1]
53      else:
54        return False
55    else:
56      if len(intersections):
57        return intersections[0]
58      else:
59        return False
60
61
62  def coord_line(dim: int, image, point1: tuple, point2: tuple, color: tuple,
63                 thickness: int):
64    return cv2.line(
65        image, (int(point1[0] * 50) + 25, dim * 50 - 25 - int(point1[1] * 50)),
66        (int(point2[0] * 50) + 25, dim * 50 - 25 - int(point2[1] * 50)), color,
67        thickness)
68
69
70  def hextorgb(hex):
71    return tuple(int(hex[i:i + 2], 16) for i in (0, 2, 4))[::-1]
72
73
74  def vectorlen(vector):
75    return np.sqrt(vector[0] * vector[0] + vector[1] * vector[1])
76
```

```
77
78  def localvector(basevec, globalvec):
79    return [globalvec[0] - basevec[0], globalvec[1] - basevec[1]]
80
81
82  def globalvector(basevec, localvec):
83    return [basevec[0] + localvec[0], basevec[1] + localvec[1]]
84
85
86  def normvector(vector):
87    return [vector[0] / vectorlen(vector), vector[1] / vectorlen(vector)]
88
89
90  def rotate(x, y, xo, yo, theta):  # rotate x,y around xo,yo by theta (rad)
91    xr = math.cos(theta) * (x - xo) - math.sin(theta) * (y - yo) + xo
92    yr = math.sin(theta) * (x - xo) + math.cos(theta) * (y - yo) + yo
93    return [xr, yr]
94
95
96  def anglesort(vector, angle):
97    newvec = rotate(vector[0], vector[1], 0, 0, angle)
98    atn = math.atan2(newvec[1], newvec[0])
99    if atn < 0:
100     atn = 2 * math.pi - abs(atn)
101   return atn
102
103
104 def angbetween(start, end, mid):
105   end = end - start + math.pi * 2 if end - start < 0 else end - start
106   mid = mid - start + math.pi * 2 if mid - start < 0 else mid - start
107   return mid < end
108
109
110 def main():
111   inputdata = list(map(float, input().split()))
112   H = int(inputdata[0])
113   N = int(inputdata[1])
114   ALPHA = inputdata[2]
115   H //= 250
116   dirdict = {"L": Dir.LEFT, "U": Dir.UP, "R": Dir.RIGHT, "D": Dir.DOWN}
117
118   inputdata = input().split()
119   D = dirdict[inputdata[0]]
120   Xs = int(inputdata[1]) / 250 - 0.5
121   Ys = int(inputdata[2]) / 250 - 0.5
122   BETA = float(inputdata[3])
123   del inputdata
124
125   borders = []
126   for i in range(N):
127     data = input().split()
128     data[:4] = list(map(lambda x: int(x) / 250 - 0.5, data[:4]))
129     data[0], data[1], data[2], data[3] = min(data[0], data[2]), min(
130         data[1], data[3]), max(data[0], data[2]), max(data[1], data[3])
131     borders.append(data)
132
133   visiblepoints = []
134   for border in borders:
135     border_xlen = border[2] - border[0]
136     border_ylen = border[3] - border[1]
```

```
    borderpointc = [(border[0] + border[2]) / 2, (border[1] + border[3]) / 2]
    borderpoint1 = [border.copy()[0], border.copy()[1]]
    borderpoint2 = [border.copy()[2], border.copy()[3]]
    borderpoint3 = [border.copy()[0], border.copy()[1]]
    borderpoint4 = [border.copy()[2], border.copy()[3]]
    if border_xlen:
      borderpoint1[0] += 0.001
      borderpoint2[0] -= 0.001
      borderpoint3[0] -= 0.001
      borderpoint4[0] += 0.001
    if border_ylen:
      borderpoint1[1] += 0.001
      borderpoint2[1] -= 0.001
      borderpoint3[1] -= 0.001
      borderpoint4[1] += 0.001
    locvec1 = localvector((Xs, Ys), borderpoint3)
    locvec1[0] *= 100
    locvec1[1] *= 100
    raycastvec1 = globalvector((Xs, Ys), locvec1)
    locvec2 = localvector((Xs, Ys), borderpoint4)
    locvec2[0] *= 100
    locvec2[1] *= 100
    raycastvec2 = globalvector((Xs, Ys), locvec2)

    res = raycast((Xs, Ys), borderpoint1, borders)
    if res:
      image = coord_line(H, image, (Xs, Ys), (res[0][0], res[0][1]),
                         hextorgb(res[1]), 1)
      print(res)
      visiblepoints.append(res)

    res = raycast((Xs, Ys), borderpoint2, borders)
    if res:
      image = coord_line(H, image, (Xs, Ys), (res[0][0], res[0][1]),
                         hextorgb(res[1]), 1)
      print(res)
      visiblepoints.append(res)

    res = raycast((Xs, Ys), raycastvec1, borders)
    if res:
      image = coord_line(H, image, (Xs, Ys), (res[0][0], res[0][1]),
                         hextorgb(res[1]), 1)
      print(res)
      visiblepoints.append(res)

    res = raycast((Xs, Ys), raycastvec2, borders)
    if res:
      image = coord_line(H, image, (Xs, Ys), (res[0][0], res[0][1]),
                         hextorgb(res[1]), 1)
      print(res)
      visiblepoints.append(res)

    res = raycast((Xs, Ys), borderpointc, borders)

  dirarr = [math.pi, math.pi / 2, 0, -math.pi / 2]

  leftbound = ALPHA / 2
  rightbound = -ALPHA / 2
  if D == Dir.LEFT:
    leftbound += math.pi
```

```
197       rightbound += math.pi
198     elif D == Dir.UP:
199       leftbound += math.pi / 2
200       rightbound += math.pi / 2
201     elif D == Dir.RIGHT:
202       leftbound += 0
203       rightbound += 0
204     elif D == Dir.DOWN:
205       leftbound += math.pi / 2 * 3
206       rightbound += math.pi / 2 * 3
207 
208     if BETA > 0:
209       leftbound += BETA
210     else:
211       rightbound -= BETA * -1
212 
213     if leftbound > rightbound:
214       pass
215     else:
216       rightbound, leftbound = leftbound, rightbound
217     endray = []
218     if BETA > 0:
219       endray = raycast(
220           (Xs, Ys),
221           globalvector((Xs, Ys), (math.cos(leftbound - 0.001) * 10000,
222                                   math.sin(leftbound - 0.001) * 10000)), borders)
223     elif BETA < 0:
224       endray = raycast((Xs, Ys),
225                        globalvector((Xs, Ys),
226                                     (math.cos(rightbound + 0.001) * 10000,
227                                      math.sin(rightbound + 0.001) * 10000)),
228                        borders)
229     if endray:
230       pass
231       endray = list(endray)
232       endray.append("E")
233       visiblepoints.append(endray)
234     else:
235       if BETA > 0:
236         angi = leftbound - 0.001
237         while angi > rightbound + 0.001:
238           endray = raycast(
239               (Xs, Ys),
240               globalvector((Xs, Ys),
241                            (math.cos(angi) * 10000, math.sin(angi) * 10000)),
242               borders)
243           if endray:
244             endray = list(endray)
245             endray.append("E")
246             visiblepoints.append(endray)
247             break
248           else:
249             angi -= 0.001
250       elif BETA < 0:
251         angi = rightbound + 0.001
252         while angi < leftbound - 0.001:
253           endray = raycast(
254               (Xs, Ys),
255               globalvector((Xs, Ys),
256                            (math.cos(angi) * 10000, math.sin(angi) * 10000)),
```

```
257             borders)
258         if endray:
259           endray = list(endray)
260           endray.append("E")
261           visiblepoints.append(endray)
262           break
263         else:
264           angi += 0.001
265 
266   visiblepoints.sort(key=lambda x: anglesort(localvector(
267       (Xs, Ys), x[0]), -dirarr[D] + ALPHA / 2))
268   if BETA < 0:
269     visiblepoints.reverse()
270   startray = []
271   if BETA > 0:
272     startray = raycast((Xs, Ys),
273                        globalvector((Xs, Ys),
274                                     (math.cos(rightbound + 0.001) * 10000,
275                                      math.sin(rightbound + 0.001) * 10000)),
276                        borders)
277   elif BETA < 0:
278     startray = raycast(
279         (Xs, Ys),
280         globalvector((Xs, Ys), (math.cos(leftbound - 0.001) * 10000,
281                                 math.sin(leftbound - 0.001) * 10000)), borders)
282   if startray:
283     pass
284     visiblepoints.insert(0, startray)
285   i = 0
286   vecs = []
287 
288   markremoval = []
289   i = 0
290   for i in range(len(visiblepoints)):
291     if visiblepoints[i][-1] == "E":
292       locvec1 = ()
293       if BETA > 0:
294         locvec1 = (math.cos(rightbound), math.sin(rightbound))
295       else:
296         locvec1 = (math.cos(leftbound), math.sin(leftbound))
297       locvec2 = localvector((Xs, Ys), visiblepoints[i][0])
298       normvec1 = normvector(locvec1)
299       normvec2 = normvector(locvec2)
300       anglebetween = np.arccos(np.dot(normvec1, normvec2))
301       if abs(
302           BETA) <= math.pi or abs(BETA) > math.pi and anglebetween > ALPHA * 2:
303         visiblepoints = visiblepoints[:i + 1]
304       break
305   i = 0
306   while i < len(visiblepoints) - 1:
307     locvec1 = localvector((Xs, Ys), visiblepoints[i][0])
308     locvec2 = localvector((Xs, Ys), visiblepoints[i + 1][0])
309     normvec1 = normvector(locvec1)
310     normvec2 = normvector(locvec2)
311     anglebetween = np.arccos(np.dot(normvec1, normvec2))
312     angle1 = math.atan2(normvec1[1], normvec1[0])
313     angle2 = math.atan2(normvec2[1], normvec2[0])
314     locvec3 = None
315     normvec3 = None
316     angle3 = None
```

```
317       if i < len(visiblepoints) - 2:
318         locvec3 = localvector((Xs, Ys), visiblepoints[i + 2][0])
319         normvec3 = normvector(locvec3)
320         angle3 = math.atan2(normvec3[1], normvec3[0])
321       if i < len(visiblepoints) - 2 and visiblepoints[i][1] == visiblepoints[
322           i + 1][1] and visiblepoints[i][1] == visiblepoints[i + 2][1] and (
323               visiblepoints[i][2] == visiblepoints[i + 1][2]
324               or abs(angle2 - angle1) < 0.005) and (
325                   visiblepoints[i + 1][2] == visiblepoints[i + 2][2]
326                   or abs(angle3 - angle2) < 0.005):
327         markremoval.append(i + 1)
328         i += 1
329       else:
330         i += 1
331     for i in markremoval:
332       visiblepoints[i] = 0
333     i = 0
334     while i < len(visiblepoints):
335       if visiblepoints[i] == 0:
336         visiblepoints.pop(i)
337       else:
338         i += 1
339     for i in range(len(visiblepoints) - 1):
340       if visiblepoints[i][1] == visiblepoints[i + 1][1]:
341         locvec1 = localvector((Xs, Ys), visiblepoints[i][0])
342         locvec2 = localvector((Xs, Ys), visiblepoints[i + 1][0])
343         normvec1 = normvector(locvec1)
344         normvec2 = normvector(locvec2)
345         anglebetween = np.arccos(np.dot(normvec1, normvec2))
346         angle1 = math.atan2(normvec1[1], normvec1[0])
347         angle2 = math.atan2(normvec2[1], normvec2[0])
348         color = visiblepoints[i][1]
349         if anglebetween > ALPHA / 10:
350           vecs.append([angle1, angle2, color, anglebetween])
351 
352     colors = []
353     for i in vecs:
354       if not colors or colors[-1] != i[2]:
355         colors.append(i[2])
356     if len(colors) > 1:
357       print(' '.join(colors))
358     else:
359       if len(colors):
360         print(colors[0])
361 
362 
363 if __name__ == "__main__":
364     main()
```

Задача II.1.1.5. Калибровка датчика (10 баллов)

На робототехническом устройстве имеется датчик, который необходимо откалибровать.

Калибровка происходит следующим образом:

1. Имеется N точных значений данной величины y_i;
2. Имеется N измерений той же величины датчиком $\hat{y}_i$ в такой же последовательности;

3. $i \in [1, N]$.

Необходимо откалибровать датчик так, чтобы он давал удовлетворительное (в пределах ± 5) относительное расхождение между истинным значением измеряемой величины и показаниями калиброванного датчика в M промежуточных точках.

Для калибровки можно использовать различные методы интерполяции (сплайны).

Рассмотрим две из них:

1. Кусочно-линейная интерполяция — на каждом отрезке функция приближается линейной, других условий не требуется;
2. Гладкая кусочно-кубическая интерполяция — на каждом отрезке функция приближатся кубическим многочленом, дополнительно требуется непрерывность первой и второй производных функции на всём отрезке.

Кусочно-линейная интерполяция:

$$U(x) = s_i(x); \tag{II.1.6}$$

$$x \in [x_{i-1}, x_i]; \tag{II.1.7}$$

$$s_i(x) = u(x_{i-1})\frac{x_i - x}{x_i - x_{i-1}} + u(x_i)\frac{x - x_{i-1}}{x_i - x_{i-1}} \tag{II.1.8}$$

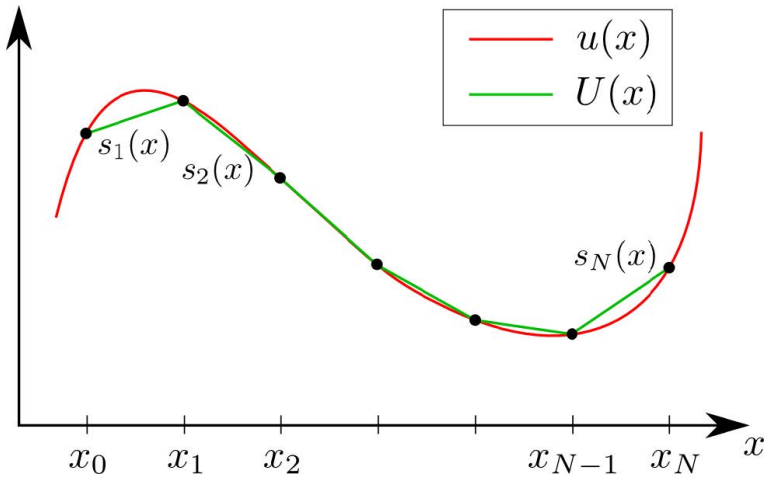

Рис. II.1.14: Кусочно-линейная интерполяция

Гладкая кусочно-кубическая интерполяция:

$$U(x) = s_i(x); \tag{II.1.9}$$

$$x \in [x_{i-1}, x_i]; \tag{II.1.10}$$

$$s_{i-1}(x_{i-1}) = s_i(xi - 1); \tag{II.1.11}$$

$$s_i(x_i) = s_{i+1}(xi); \tag{II.1.12}$$

$$s'_{i-1}(x_{i-1}) = s'_i(xi - 1); \tag{II.1.13}$$

$$s'_i(x_i) = s'_{i+1}(xi); \tag{II.1.14}$$

$$s''_{i-1}(x_{i-1}) = s''_i(xi - 1); \tag{II.1.15}$$

$$s''_i(x_i) = s''_{i+1}(xi); \tag{II.1.16}$$

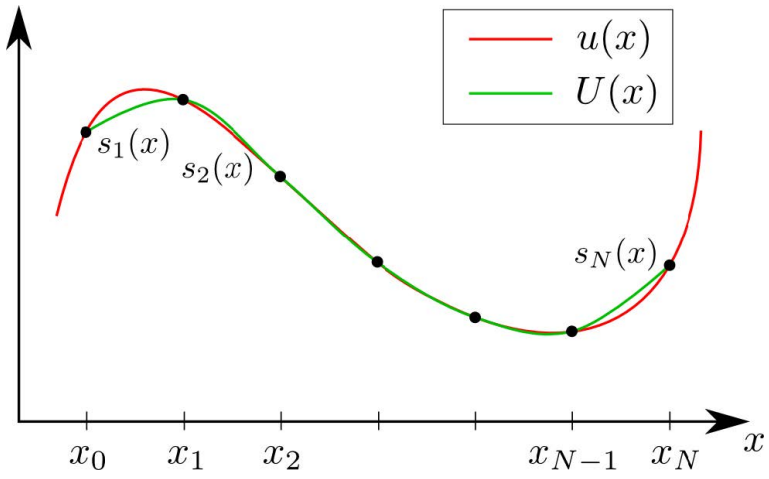

Рис. II.1.15: Кусочно-линейная интерполяция

Формат входных данных

Первая строчка содержит два целых числа через пробел – N, M, где:

- N — количество проведенных тестовых измерений ($1 \leqslant N \leqslant 100$);
- M — количество промежуточных точек ($1 \leqslant M < N$).

Вторая строчка содержит N вещественных чисел через пробел — точные значения y_i измеряемой величины ($-10000 \leqslant y_i \leqslant 10000$).

Третья строчка содержит N вещественных чисел через пробел — калибровочные измерения проведенные датчиком $\hat{y}_i$ ($-10000 \leqslant \hat{y}_i \leqslant 10000$).

Последняя строчка содержит M вещественных чисел через пробел — измерения проведенные датчиком $\hat{x_j}$ ($min(\hat{y}_i) \leqslant \hat{x_j} \leqslant max(\hat{y}_i)$).

Формат выходных данных

Одна строка, содержащая M вещественных чисел через пробел — откалиброванные измерения, проведенные датчиком x_i.

Примеры

Примеры входных данных и ответов к ним можно найти по данной ссылке.

Решение

Рассмотрим пример решения задачи методом кусочно-линейной интерполяции (далее - КЛИ).

Интерполяция - метод нахождения приближенной или точной величины по известным отдельным значениям этой величины (или можно сказать - восстановление функции по нескольким известным величинам).

Кусочно-линейная интерполяция относится к локальному методу интерполяции (когда на каждом интервале $[x_{i-1}, x_i]$ строим свою 'локальную' функцию).

По условиям задачи нам дается три набора данных:

1. Точные значения данной величины (*precision*)
2. Калибровочные измерения, проведенные датчиком (*calibrate*)
3. Измерения, проведенные датчиком (*values*)

На каждом отрезке заменяем функцию линией $F_i(x) = k_i x + b_i$. Для этого нам надо найти коэффициенты k_i и b_i. Условия кусочно-линейной интерполяции: $F_i(x_{i-1}) = f_{i-1}$ и $F_i(x_i) = f_i$, где f - локальные функции, где $k_i x_{i-1} + b_i = f_{i-1}$ и $k_i x_i + b_i = f_i$. Исходя из этого, находим коэффициенты по формулам II.1.17:

$$k_i = \frac{f_i - f_{i-1}}{x_i - x_{i-1}} \qquad b_i = f_i - k_i x_i \tag{II.1.17}$$

Далее находим, в какой интервал (x_{i-1}, x_i) попадает искомая точка x. Вычисляем коэффициенты k_i и b_i и находим значение $F(x)$

Для примера возьмем набор данных N3 и найдем $F(x)$ для значения 8.289400292 $(VALUE)$, это второй элемент в массиве *values*.

Сначала находим значения по оси X в интервале которых находится значение $VALUE$. На графике II.1.16 видно, что значение $VALUE$ находится между 7.917250.. и 8.326198... Для точного результата нужно проверять значения по всем данным (на примере график показан по первым 20 значениям).

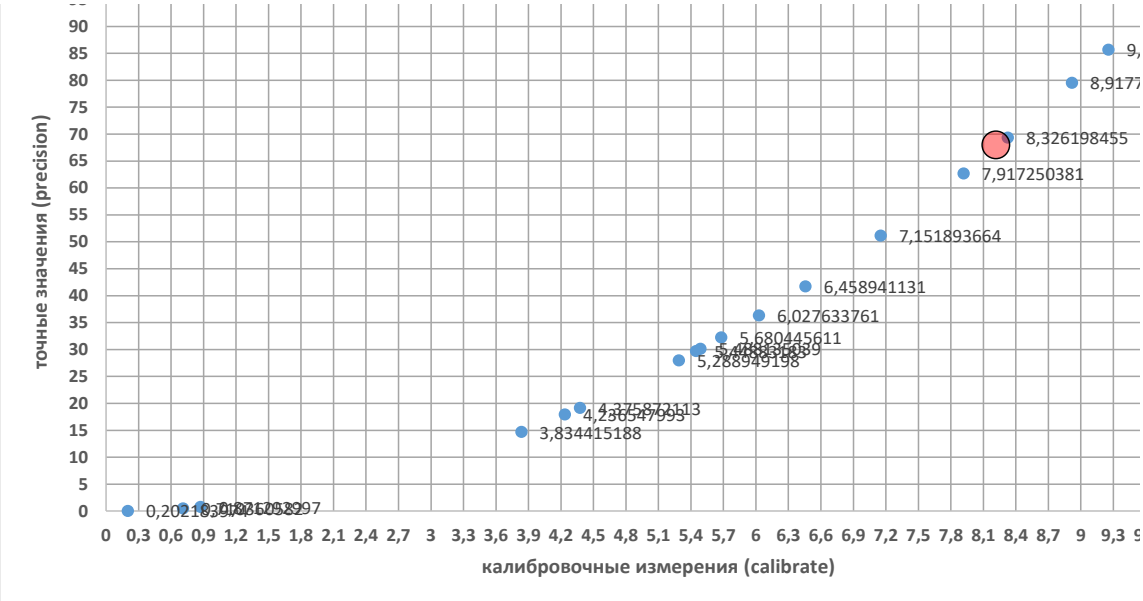

Рис. II.1.16: График по первым 20 значениям (пример)

Во время определения ближайших чисел необходимо запоминать их порядковые индексы в общем массиве *calibrate*. Далее по этим индексам мы значения из массива точных значений *precision*.

В нашем случае, это индексы [10] и [17] и, соответственно, значения из массива *precision* - 62.68285359 и 69.32558072 (см. рис. II.1.17)

idx	precision	calibrate	values
0	30,119626	5,488135	**8,2894003**
1	51,149583	7,1518937	0,0469548
2	36,332369	6,0276338	6,7781654
3	29,689768	5,4488318	2,7000797
4	17,948339	4,236548	7,3519402
5	41,717921	6,4589411	9,6218855
6	19,148257	4,3758721	2,4875314
7	79,525908	8,91773	5,7615733
8	92,864592	9,6366276	5,9204193
9	14,70274	3,8344152	5,7225191
10	**62,682854**	**7,9172504**	2,2308163
11	27,972984	5,2889492	9,5274901
12	32,267462	5,6804456	4,4712538
13	85,672914	9,2559664	8,4640867
14	0,5046122	0,7103606	6,9947928
15	0,7591515	0,871293	2,9743695
16	0,0408784	0,202184	8,1379782
17	**69,325581**	**8,3261985**	3,9650574
18	60,552793	7,7815675	8,811032
19	75,692114	8,7001215	5,8127287

Рис. II.1.17: Значения массивов данных с индексами

Теперь мы можем вычислить значение y для нашего $VALUE$:

$x_1 = 7.91725, y_1 = 62.68285 \qquad x_2 = 8.32619, y_2 = 69.32558$

$$k = \frac{y_2 - y_1}{x_2 - x_1} = \frac{69.32558 - 62.68285}{8.32619 - 7.91725} = 16.24$$

$$b = y_1 - x_1\dot{k} = 62.68285 - 7.91725 \cdot 16.24 = 62.9206$$

$$y = kx + b = 16.24 \cdot 8.289400292 + 62.9206 = 68.699$$

Результат вычисления показан на рис. II.1.18:

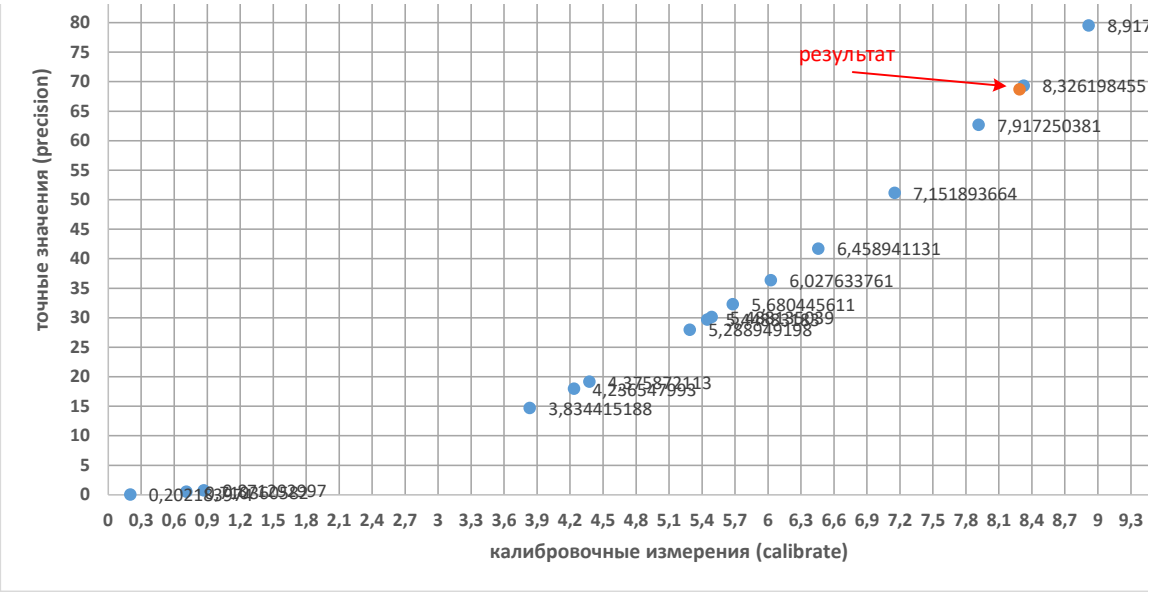

Рис. II.1.18: Результат вычисления интерполяции F(8.2894)

Аналогичным образом обрабатываем все значения из массива *values* и в качестве результата выводим полученные значения через пробел.

Пример программы-решения

Ниже представлено решение на языке Python 3

```
# -*- coding: utf-8 -*-
import sys


def solution(data):
  def getIdx(num):
    for i in range(1, len(dictX)):
      x1, x2 = dictX[i - 1], dictX[i]
      if x2[1] >= num >= x1[1]:
        return int(x1[0]), int(x2[0])

  n, m = map(int, data[0].strip().split())
  precision = list(map(float, data[1].strip().split()))
  calibrate = list(map(float, data[2].strip().split()))
  values = list(map(float, data[3].strip().split()))

  res = []

  for X in values:
    idx1, idx2 = -1, -1
    for i in range(len(calibrate)):
      if idx1 == -1 or idx2 == -1 or abs(calibrate[idx1] -
                                         X) > abs(calibrate[i] - X):
        idx2 = idx1
        idx1 = i

#  print (X, idx1, idx2)
```

```
28      x1, x2 = calibrate[idx2], calibrate[idx1]
29      y1, y2 = precision[idx2], precision[idx1]
30      a = (y2 - y1) / (x2 - x1)
31      b = y1 - x1 * a
32      y = a * X + b
33      res.append(y)
34    return res
35
36  data = sys.stdin.readlines()
37  print(*solution(data))
```

Задача II.1.1.6. Измерение расстояний (5 баллов)

Перед запуском робот, оснащенный двумя инфракрасными датчиками расстояния, установлен около стены так, что она находится слева. Один датчик расстояния направлен на стену и смонтирован так, что край левого колеса и передняя кромка датчика находятся на одной прямой. Второй датчик расстояния направлен по ходу движения робота, передняя кромка датчика не выходит за передний край корпуса робота.

В стене есть впадины разной глубины. Напишите программу для поиска самой глубокой впадины и заезда в нее. Гарантируется, что существует единственное решение. После заезда во впадину, робот должен остановиться и вывести на экран слово «finish».

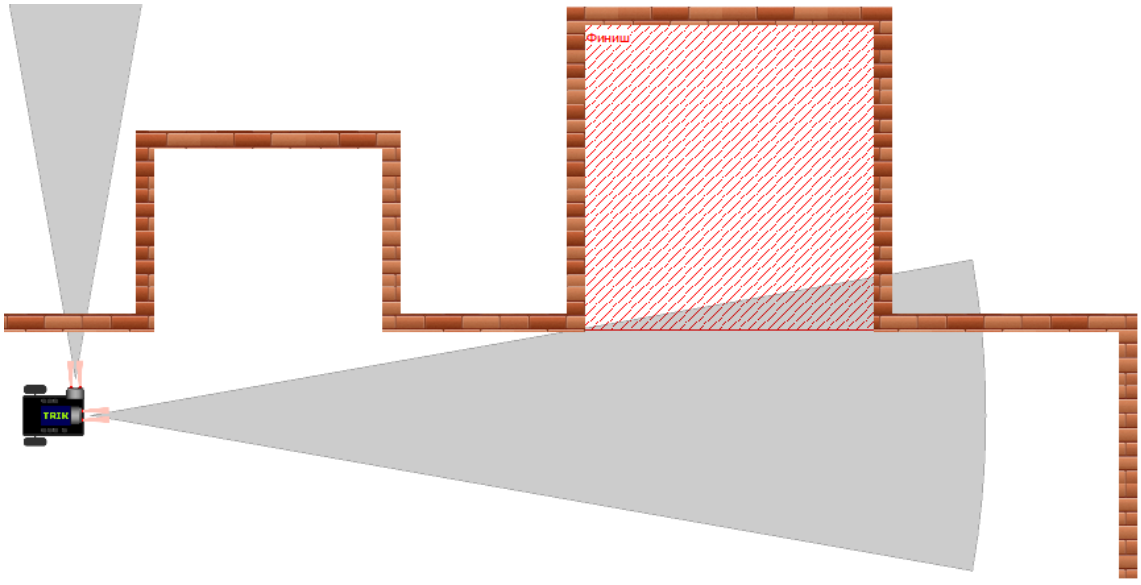

Рис. II.1.19: Внешний вид одного из вариантов поля

Конфигурация робота

Подключение моторов:

- Левый мотор — порт M3
- Правый мотор — порт M4

Подключение датчиков:

- Датчик расстояния, направленный вперед - порт D1

- Датчик расстояния, направленный влево - порт D2

Комментарии

Количество впадин может изменяться от 2 до 8. Расстояние между двумя соседними впадинами не будет меньше 525 мм. Ширина впадины - от 700 до 1050 мм. Глубина впадин может изменяться от 350 до 1000 мм, относительно передней кромки стены. От дальнего (относительно движения робота) края самой последней впадины на расстоянии 700 мм установлено препятствие.

На старте робот установлен так, что его датчик не направлен на впадину, и все впадины находятся по ходу движения робота.

Первоначальное расстояние от робота до стены может быть в диапазоне от 175 до 525 мм.

Будет считаться, что робот заехал во впадину, если самая крайняя часть робота не выступает за кромку стены.

Необходимо загружать js файл с программой, написанной в TRIK Studio 2019.7.

Примеры

Примеры тестовых полигонов представленны по данной ссылке.

Решение

Декомпозиция решения задачи:

1. Двигаться прямолинейно до стены спереди и считывать данные:
   1.1. Глубину каждой впадины
   1.2. Ширину каждой впадины
2. Определить впадину с максимальной глубиной
3. Развернуться и доехать до середины выбранной впадины
4. Повернуть и заехать в зону впадины

Прямолинейное движение и точные повороты роботов на дифференциальной платформе являются базовыми алгоритмами и данном решении не будут подробно описываться.

В случае решения задачи в симуляторе TRIK Studio для прямолинейного движения и точных поворотов лучше использовать пропорциональный регулятор с управлением по гироскопическому датчику.

Для примера возьмем карту N11 из набора данных.

Перед началом движения робот запоминает расстояние до стены слева. Это будет являться для него “0-уровнем” отметки ($wall$). После начала движения ждем увеличения расстояния слева на бОльшее, чем $wall$. При увеличении расстояния запоминаем значения энкодеров в данной точке (V_{ix0}). При считывании глубины впадины нужно учитывать расстояние до стены:

$$depth_i = sensorDistance - wall$$

где **sensorDistance** - показания датчика расстояния, а **wall** - расстояние до стены при старте

Робот продолжает движение и ждет (слева по ходу движения) расстояния, равное *wall* с погрешностью 1..2 см. После чего снова запоминаем значение энкодеров в данной точке (V_{ix1}).

Повторяем операции для каждой впадины, пока робот не доедет до стены спереди (когда расстояние датчика спереди не будет меньше, чем 15 см).

Когда робот доезжает до стены спереди по ходу движения - останавливаем робота.

На рис. II.1.20 показаны данные, которые у нас есть на момент остановки робота.

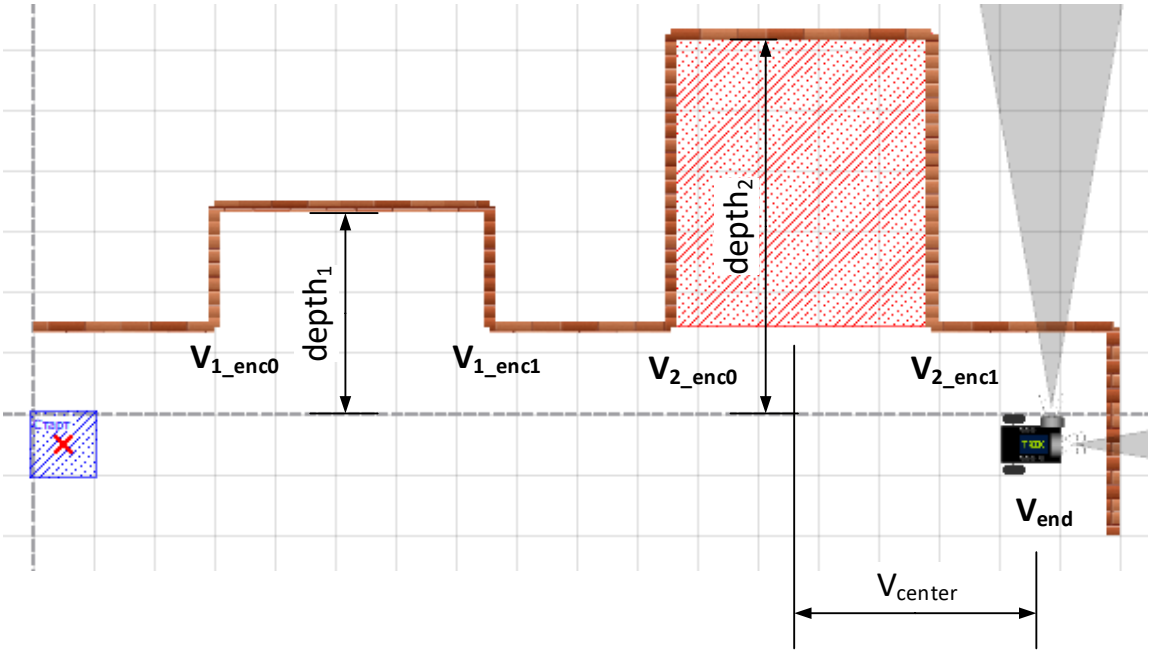

Рис. II.1.20: Полученные данные на момент остановки робота

Определяем впадину с максимальной глубиной, поочередно сравнивая все значения $depth_i$.

Определяем расстояние до центра выбранной впадины в “считываниях энкодера (тиках)” (V_{center}) по формуле:

$$V_{center} = V_{end} - \frac{V_{ix0} + V_{ix1}}{2}$$

После этого робот поворачивается на 180° и движется прямо на расстояние, равное V_{center} тиков энкодера относительно текущих значений энкодеров.

Робот доезжает до середины впадины. Разворачивается вправо на 90°. Едет прямо пока расстояние спереди не станет меньше 15 см. Выводит на экран надпись **“finish”**.

Пример программы-решения

Ниже представлено решение на языке JavaScript

```
function motors(vL, vR){
        vL = vL || 90, vR = vR || vL
        brick.motor('M4').setPower(vL)
        brick.motor('M3').setPower(vR)
}


function turnGyro(angle){
        angle *= 1000
        motors(40, -40)
        while (angle - yaw() > 2000) script.wait(20)
        motors(0)
}


function goGyroCpr(goal, gyro0){
        gyro0 *= 1000
        goal += eL()
        while (eL() < goal){
                var y = yaw()
                u = ((gyro0 - abs(y)) * sign(y))/1000 * 0.7
                motors(80 + u, 80 - u)
                script.wait(20)
        }
        motors(0)
}


function goFront(gyro0, getData){
        gyro0 *= 1000
        while (sFront() > 20){
                if (getData){
                        raw.push(sLeft())
                        encs.push(eL())
                }
                u = (gyro0 - yaw())/1000 * 0.7
                motors(80 + u, 80 - u)
                script.wait(20)
        }
}

//
function printDebug(info,txt){txt = txt|| ''; if (debug) print (txt, info)}
function yaw(){ return brick.gyroscope().read()[6]}
function sign(num){ return num > 0 ? 1 : -1}
function inRange(num, range){return num >= range[0] && num <= range[1]}

sFront = brick.sensor('D1').read
sLeft  = brick.sensor('D2').read
eL = brick.encoder('E4').read
eR = brick.encoder('E3').read
wait = script.wait
abs = Math.abs

brick.gyroscope().calibrate(2000)
wait(2100)
```

```
57
58  raw = [], encs = []
59  wall = sLeft()
60
61  goFront(0, true)
62
63  max = Math.max.apply(null, raw)
64  min = Math.min.apply(null, raw)
65
66
67  maxData = [-1, 0, 0]
68  for (var i=0, next=false, start=0, number=0; i < raw.length; i++){
69          if (raw[i] > wall + 5 && !next) next = true, number++, start = encs[i]
70          if (raw[i] == max) maxData[0] = number, maxData[1] = start
71          if (raw[i] < wall + 5){
72                  if (maxData[0] == number && next) maxData[2] = encs[i]
73                  next = false, start = 0
74          }
75  }
76
77
78  goal = eL() - ((maxData[1] + maxData[2])/2)-275
79  turnGyro(180)
80  goGyroCpr(goal, 180)
81
82  turnGyro(-90)
83  goFront(-90)
84
85  brick.display().addLabel('finish', 1, 1)
86  brick.display().redraw()
```

Задача II.1.1.7. Перемещение в лабиринте (10 баллов)

Робот, собранный по дифференциальной схеме оснащен тремя инфракрасными датчиками расстояния. Один из датчиков расстояния установлен так, что показывает расстояние до препятствий прямо по курсу робота. Второй датчик расстояния направлен влево. Третий датчик расстояния направлен вправо.

Необходимо в заранее неизвестном лабиринте размером 8×8 секторов переместиться в сектор финиша. Сектор — квадрат шириной 700 мм. Координаты сектора финиша передаются в виде двух изображений ArTag маркеров через входной файл.

Началом координат служит верхний левый сектор лабиринта. Положительным направлением оси X считается направление по горизонтали вправо. Положительным направлением оси считается направление по вертикали вниз. Расположение осей координат представлено на рис. II.1.21. Гарантируется что при старте робот направлен в сторону положительного направления оси X.

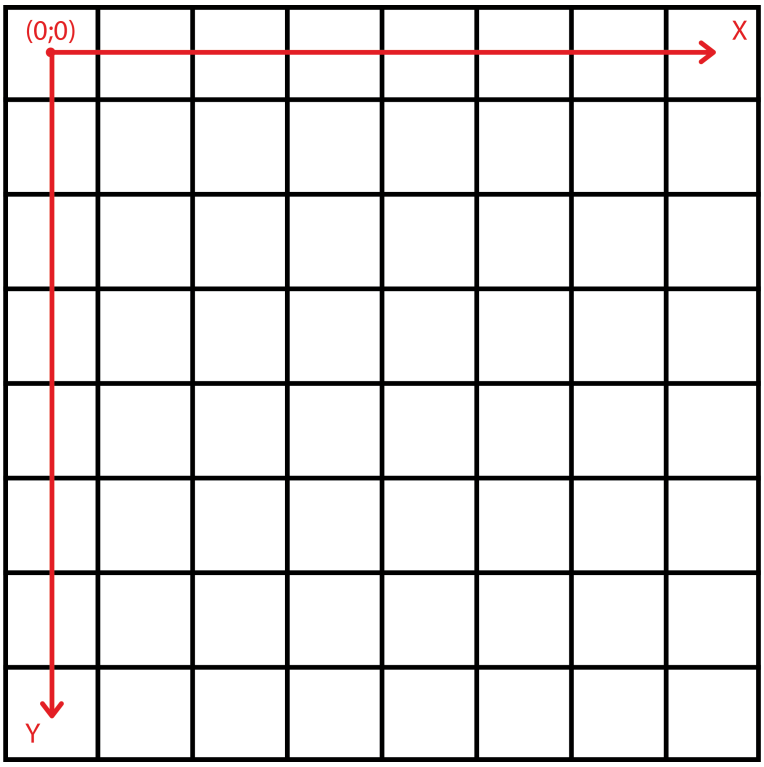

Рис. II.1.21: Расположение осей координат

Изображение поверхности с ArTag маркером приходит в управляющую программу в виде набора 160 × 120 точек, где каждая точка закодирована в RGB-формате.

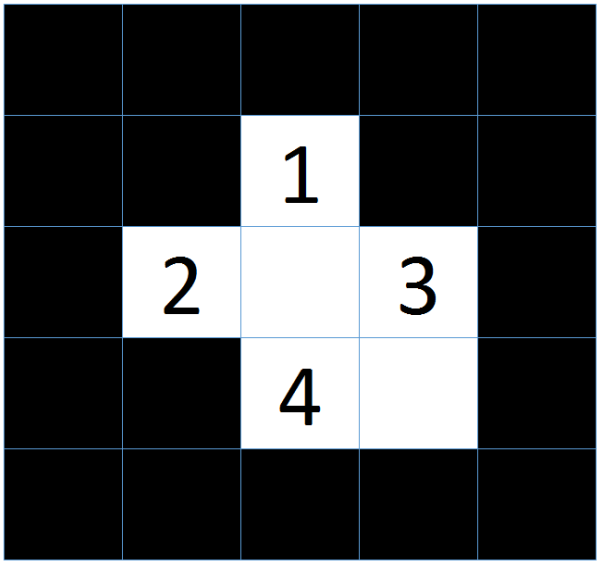

Рис. II.1.22: Нумерация битов в маркера

Элементы ARTag маркеры (https://inside.mines.edu/~whoff/courses/EENG512/lectures/other/ARTag.pdf), расположенные по его границе — всегда черные. Четыре элемента, находящиеся в углах внутреннего 3 × 3 квадрата определяют ориентацию маркера таким образом, что только элемент в нижнем правом углу квадрата — белый. Центральный элемент квадрата используется для проверки четности (parity check): если количество единичных бит в двоичной записи закодированного в маркере числа нечетное, то он черный. Оставшиеся 4 элемента маркера кодируют число

по следующему правилу: если элемент черный, то он обозначает 1, если белый, то 0 при этом самый первый элемент — старший бит закодированного числа. Элементы пронумерованы сверху вниз, слева направо (см. рис. II.1.22).

В случае если число меньше 8, то данные маркер кодирует координату по оси X. В противном случае следует вычесть 8 из полученного значения и получим координату по оси Y.

Например, на маркере с рис. II.1.23 закодировано число 0011_2, что эквивалентно 3_{10}. Следовательно, это X составляющая равная 3.

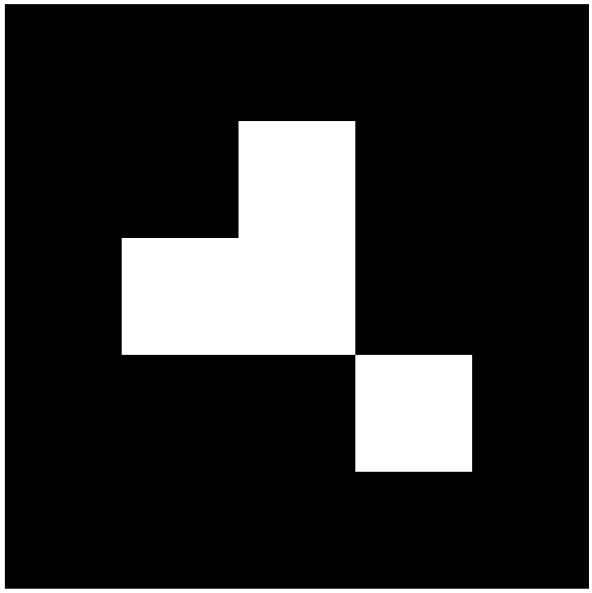

Рис. II.1.23: Маркер с закодированным значением - 0011_2

ris:arTagExample

Поскольку изображение было получено в процесе движения, то изображения ArTag маркера, получаемые с камеры, получаются в виде неправильного выпуклого четырехугольника, а непостоянные условия освещенности изменяют фокус и тон на изображении. Ориентация маркеров также заранее неизвестна, но его изображение таково, что оно по каждой из осей X, Y, Z относительно оптической оси камеры не превышает 25 градусов.

Формат входных данных

Входной файл содержит 2 строки, на каждой расположено изображение размером 160×120 в виде шестнадцатеричных чисел слева направо сверху вниз. Данные числа имеют следующий вид: $RRGGBB$, где RR - 16тиричное число R составляющей данного элемента матрицы, GG и BB — эт0 16ти-ричные числа G и B составляющих соответственно.

Все числа являются целыми.

Формат выходных данных

Доехать до сектора финиш и вывести на экран слово «finish».

Конфигурация робота

Подключение моторов:

- Левый мотор — порт M3
- Правый мотор — порт M4

Подключение датчиков:

- Датчик расстояния, направленный вперед - порт A1
- Датчик расстояния, направленный вправо - порт A2
- Датчик расстояния, направленный влево - порт A3

Примеры

Примеры тестовых полигонов представленны по данной ссылке.

Решение

Декомпозиция решения задачи:

1. Декодирование метки дополненной реальности ArTag
   1.1. Считать данные из файлов
   1.2. Распознать метки (получить десятичное число)
   1.3. Определить координаты X и Y сектора финиша
2. Исследовать неизвестный лабиринт и составить карту
3. Определить собственные координаты и направление (локализоваться)
4. Переместиться в сектор финиша

Распознавание меток ArTag подробно описано в материалах Университета Иннополис в разделе "Программирование интеллектуальных робототехнических систем" по адресу: Подраздел “Алгоритмические основы технического зрения”

Прямолинейное движение и точные повороты роботов на дифференциальной платформе являются базовыми алгоритмами и данном решении не будут подробно описываться.

Для точной навигации удобнее использовать П-регуляторы с управлением по энкодерам и гироскопу. Некоторые участники еще используют стенки лабиринта для выравнивания робота.

Согласно условиям задачи, робот стартует в заранее неизвестном секторе в направлении “восток (направо от начала координат)”.

Так как нам известен размер лабиринта - 8x8 секторов, но не известен стартовый сектор, представим весь лабиринт в размере 15x15 секторов. Предположим, что робот стартует в самом центре, в секторе с координатами $X : 7, Y : 7$. Двигаясь в любом направлении из этого сектора, робот сможет проехать максимум 8 секторов.

Возможность перемещения по секторам робот определяет с помощью датчиков расстояния. Перемещаясь между секторами робот составляет карту лабиринта в виде матрицы смежности, которая содержит информацию о возможности или не возможности перемещения в смежные сектора из текущего сектора.

Можно учитывать “тип” секторов - секторы в которых робот проезжал и секторы, в которые есть возможность проезда, но робот их не посещал.

Планирование и навигацию между секторами, особенно в случае, когда робот в месте, где с трех сторон стенки или все смежные сектора уже посещены, можно осуществлять с помощью алгоритма BFS (поиск в графе в ширину), составляя пути перемещения робота.

После исследования всей карты лабиринта определяем минимальные координаты реального лабиринта по осям X и Y ($deltaX$ и $deltaY$) и далее используем их для вычисления координат секторов для выполнения задания.

Расчет пути движения робота в сектор финиша выполнять через алгоритмы обхода графов: BFS, Дейкстры или A-star. Использование алгоритмов "правой руки" или "левой руки" в данном случае может не помочь, т.к. лабиринты могут быть "зациклены".

Пример программы-решения

Ниже представлено решение на языке JavaScript

```
function sign(num) {return num >= 0 ? 1 : -1}
function getYaw() {return brick.gyroscope().read()[6]}
function cm2cpr(cm) {return (cm / (Math.PI * robot.wheelD)) * robot.cpr}
function rad2deg(rad) {return rad * 180 / Math.PI }
function deg2rad(deg) {return deg * Math.PI / 180 }
function inRange(value, range){return (value >= range[0] && value <= range[1]) }
function printObj(obj){for (key in obj) print(key, ': ', obj[key]) }


function arrSum(arr){
        for(var i=0, summ=0, maxI = arr.length; i < maxI; i++)
                summ += arr[i].reduce(function(a, b) {return (a + b)})
        return summ
}

function arr1arr2(arr, cols, rows){
        var arrOut = []
        for (var i=0; i < rows; i++){
                arrOut.push([])
                for (var j=0; j < cols; j++)
                        // arrOut[i].push(arr[i * cols +
                        ↪  j])                                  // HEX
                        arrOut[i].push(parseInt(arr[i * cols + j],
                        ↪  16))                 // DEC
        }
        return arrOut
}

function printArr(arr, delim, file){
        delim = delim == undefined ? ',': delim

        if (file != undefined)
                script.removeFile(file)

        for (var i=0; i< arr.length; i++){
                var line = arr[i].join(delim)
                print (line)
```

```
36
37                  if (file != undefined){
38                          script.writeToFile(file, line + '\n')
39  }}}
40
41  function artag(arr, size){
42          function parityCheck(numBin, center){
43                  var binP = picBW[center[0]][center[1]]
44                  for (var i = 0, summ=0; i < numBin.length; i++) summ += numBin[i]
45                  return summ % 2 == binP
46          }
47
48          function makeBin(arr, key){
49                  var variants = [[2,1,3,0], [3,2,0,1], [0,3,1,2], [1,0,2,3]]
50                  var arrOut = []
51                  for (var i=0; i < 4; i++) arrOut.push(arr[variants[key][i]])
52                  return arrOut
53          }
54
55          //
56          size = size || [160, 120]
57          picW = size[0], picH = size[1]
58          var picRaw = arr1arr2(arr, picW, picH)
59          picBW  = pic2bw(pic2grey(picRaw))
60
61          var ABCD1=[], ABCD2 = [], ABCD
62          ABCD1 = findCorners_1(picBW)
63          ABCD2 = findCorners_2(picBW)
64          //print (ABCD1)
65          //print ('ABCD2: ',ABCD2)
66
67          ABCD = ABCD1
68
69          if (gauss(ABCD2) > gauss(ABCD1))
70                  ABCD = ABCD2
71
72          var center = findCross(ABCD)
73          var dotsDiag = []
74          var dots = []
75
76          for (var i = 0; i < 4; i++)
77                  dotsDiag.push(lineToSegmets(center, ABCD[i], 5))
78
79          for (var i = 0; i < 4; i++)
80                  dots.push(lineToSegmets(dotsDiag[i][2], dotsDiag[(i+1)%4][2], 2))
81
82          var KEY = -1
83          for (var i=0; i < 4; i++){
84                  if (picBW[dotsDiag[i][2][0]][dotsDiag[i][2][1]] == 0){
85                          KEY = i
86                          break
87                  }
88          }
89
90          for (var i=0, nums=[]; i < 4; i++)
91                  nums.push(picBW[dots[i][1][0]][dots[i][1][1]])
92
93          var bin = makeBin(nums, KEY)
94
95          if (parityCheck(bin, center)){
```

```
                var num10 = parseInt(bin.join(''),2)
        } else {
                return -3
        }

        return num10
}

function findCorners_1(arr, delta) {
        delta = delta || 4
        var A, B, C, D, x, y, l
        var roiH = picH-1-delta, roiW = picW-1-delta

        // top-left corner
        x = y = l = delta
        while (arr[y][x] != 1) {
                if (y >= roiH)
                        l += 1, y = delta, x = l

                if (x <= delta)
                        l += 1, y = delta, x = l
                else
                        y += 1, x -= 1
        }
        A = [y, x]

        //top-right corner
        x = roiW, y = l = delta
        while (arr[y][x] != 1) {
                if (y >= picH - 1 - delta) {
                        l += 1, y = delta, x = roiW - l
                        continue
                }
                if (x >= picW - 1 - delta)
                        l += 1, y = delta, x = roiW - l
                else
                        y += 1, x += 1
        }
        B = [y, x]

        //bottom-down corner
        x = roiW, y = roiH, l = delta
        while (arr[y][x] != 1) {
                if (y <= 1 + delta) {
                        l += 1, y = roiH, x = roiW - l
                        continue
                }
                if (x >= picW - 1 - delta)
                        l += 1, y = roiH, x = roiW - l
                else
                        y -= 1, x += 1
        }
        C = [y, x]

        //bottom-left corner
        x = l = delta, y = roiH
        while (arr[y][x] != 1) {
                if (y <= 1 + delta) {
                        l += 1, y = roiH, x = l
                        continue
```

```
156                 }
157                 if (x <= delta)
158                         l += 1, y = roiH, x = l
159                 else
160                         y -= 1, x -= 1
161         }
162         D = [y, x]
163 
164         return [A, B, C, D]
165 }
166 
167 function findCorners_2(arr, delta){                         // scanning horizontal,
    ↪  vertical
168         delta = delta || 4
169         var A, B, C, D, i, j
170         var roiW = arr[0].length - delta,
171                 roiH = arr.length - delta
172 
173         top:
174         for (i = delta; i < roiH; i++){
175                 for (j = delta; j < roiW; j++){
176                         if (arr[i][j] == 1){
177                                 A = [i, j];         break top
178         }}}
179 
180         bottom:
181         for (i = roiH - 1; i > delta-1; i--){
182                 for (j = delta; j < roiW; j++){
183                         if (arr[i][j] == 1){
184                                 C = [i, j]; break bottom
185         }}}
186 
187         right:
188         for (i = roiW - 1; i > delta-1; i--){
189                 for (j = delta; j < roiH; j++){
190                         if (arr[j][i] == 1){
191                                 B = [j, i]; break right
192         }}}
193 
194         left:
195         for (i = delta; i < roiW; i++){
196                 for (j = delta; j < roiH; j++){
197                         if (arr[j][i] == 1){
198                                 D = [j, i]; break left
199         }}}
200 
201         return [A, B, C, D]
202 }
203 
204 
205 function gauss(arr){
206 
207         arr.push(arr[0])
208 
209         var S = 0
210 
211 
212 
213         for (var i = 0; i < arr.length-1; i++){
214 
```

```
215                 S += arr[i][0] * arr[i+1][1]
216
217                 S -= arr[i][1] * arr[i+1][0]
218
219         }
220
221
222
223         S = Math.abs(S) * 0.5
224
225         return S
226
227 }
228
229
230 function lineToSegmets(xy1, xy2, segments){
231         var points = [xy1]
232         var dX = (xy2[1]-xy1[1])/segments
233         var dY = (xy2[0]-xy1[0])/segments
234
235         for (var i = 1, x, y; i < segments; i++){
236                 y = Math.floor(xy1[0] + dY * i)
237                 x = Math.floor(xy1[1] + dX * i)
238                 points.push([y, x])
239         }
240         points.push(xy2)
241         return points
242
243 }
244
245 function findCross(abcd){
246         var x1,x2,x3,x4,y1,y2,y3,y4,x,y
247         x1 = abcd[0][0], y1 = abcd[0][1]
248         x2 = abcd[2][0], y2 = abcd[2][1]
249
250         x3 = abcd[1][0], y3 = abcd[1][1]
251         x4 = abcd[3][0], y4 = abcd[3][1]
252
253         x =
        ↪  -(((x1*y2-x2*y1)*(x4-x3)-(x3*y4-x4*y3)*(x2-x1))/((y1-y2)*(x4-x3)-(y3-y4)*(x2-x1)))
254         y = ((y3-y4)*(-x)-(x3*y4-x4*y3))/(x4-x3)
255
256         return [Math.floor(x), Math.floor(y)]
257 }
258
259 function clr2rgb(clr){
260         var R, G, B
261         R = (clr & 0xFF0000) >> 16
262         G = (clr & 0x00FF00) >> 8
263         B = clr & 0x0000FF
264
265         return ([R, G, B])
266 }
267
268 function rgb2clr(R, G, B){
269         if (G == undefined && B == undefined) {
270                 G = R
271                 B = R
272         }
273         return R * Math.pow(256, 2) + G * 256 + B
```

```
}

function pic2bw(arr){
        var arrOut = []
        var threshold =  arrSum(arr) / (arr.length * arr[0].length)
        for (var i=0; i<arr.length; i++){
                arrOut.push([])
                for (var j=0; j < arr[0].length; j++)
                        arrOut[i].push(arr[i][j] > threshold ? 0 : 1)
        }
        return arrOut
}

function pic2grey(arr){
        var arrOut = [], grey, avg
        for (var i=0, maxI = arr.length; i < maxI; i++){
                arrOut.push([])
                for (var j=0, maxJ = arr[0].length; j < maxJ; j++){
                        grey = clr2rgb(arr[i][j])
                        avg = floor((grey[0] + grey[1] + grey[2])/3)
                        //arrOut[i].push(rgb2clr(avg))

                        arrOut[i].push(avg)
                }
        }
        return arrOut
}

function cell2coord(cell, cols){
        cols = cols || map.size
        var row = Math.floor(cell/cols)
        var col = cell - row * cols
        return [row, col]
}

function coord2cell(row, col, cols){
        cols = cols || map.cols
        return row * cols + col
}

function gotoCell(goalCell){ doCommands(makeCommands(bfs(robot.cell, goalCell))) }
function bfs(start, end, mtrx){
        mtrx = mtrx || map.matrix
        var queue = [start]
        var visited = []
        var path = []
        for (var i=0; i < mtrx.length; i++) visited.push(0)
        while (queue.length > 0){
                var p = queue.shift()
                if (visited[p] == 0){
                        visited[p] = 1
                        path.push(p)
                        for (var i=0; i<mtrx.length; i++){
                                if (mtrx[p][i] == 1 && visited[i] == 0)
                                        queue.push(i)
                        }
                }
                if (p == end) break
        }
        var pathBack = [Number(path.slice(-1))]
```

```
334         path.reverse()
335         for (var i = 1; i < path.length; i++){
336                 if (mtrx[pathBack.slice(-1)][path[i]] == 1)
337                         pathBack.push(path[i])
338         }
339         pathBack.reverse()
340         return pathBack
341 }
342
343 function makeCommands(cells, dir){
344
345         dir = dir || robot.dir
346         var commands = []
347         var sides = [-map.size, 1, map.size, -1]
348         var actions = [         ['F', 'L', 'LL', 'R'],          // -map.size
349                                 ['R', 'F', 'L', 'LL'],          // 1
350                                 ['LL', 'R', 'F', 'L'],          // map.size
351                                 ['L', 'LL', 'R', 'F']           // -1
352                               ]
353         for (var i = 1; i < cells.length; i++){
354                 var nextDir = sides.indexOf(cells[i] - cells[i-1])
355                 var act = actions[nextDir][dir]
356                 dir = nextDir
357                 commands.push(act)
358                 if (act != 'F') commands.push('F')
359         }
360         return commands.join('').split('')
361 }
362
363 function doCommands(commands){
364         for (var j=0; j < commands.length; j++){
365                 if(commands[j]=="F")                    moveGyro(map.cellLength)
366                 else if(commands[j]=="R")       turnGyro(1)
367                 else if(commands[j]=="L")       turnGyro(-1)
368                 else if(commands[j]=="B")       turnGyro(2)
369         }
370 }
371
372 function motors(mL, mR){
373         if (mL == 0 && mR == undefined)
374                 mL = 0, mR = 0
375         else{
376                 mL = mL || robot.v
377                 mR = mR == undefined ? mL : mR
378         }
379         brick.motor('M4').setPower(mL)
380         brick.motor('M3').setPower(mR)
381 }
382
383 function moveGyro(path, gyro0){
384         gyro0 = gyro0 || gyroAngles[robot.dir]
385         gyro0 *= 1000
386         path = cm2cpr(path) + encoderL()
387
388         while (encoderL() < path){
389                 var y = getYaw()
390                 u = (gyro0 - y) / 1000 * 0.9
391                 if (gyro0 == 180000) u = ((gyro0 - abs(y)) * sign(y))/1000 * 0.7
392
393                 motors(robot.v + u, robot.v - u)
```

```
394                 script.wait(30)
395         }
396         motors(0)
397
398
399         switch (robot.dir){
400                 case 0: robot.y--; break
401                 case 1: robot.x++; break
402                 case 2: robot.y++; break
403                 case 3: robot.x--; break
404         }
405         robot.updateCell()
406 }
407
408 function forward(dlina, turn){
409         var L = dlina / (Math.PI * robot.wheelD)
410         var sgn = sign(dlina)
411         var err, u, vL, vR, encL, encR
412
413         L *= robot.cpr, L += encoderL()
414         encL = encoderL(), encR = encoderR()
415
416         if (sgn == 1){
417                 while (encoderL() <= L){
418                         err = (encoderL() - encL) - (encoderR() - encR)
419                         u = err * 2
420                         vL = (60 - u) * sgn
421                         vR = (60 + u) * sgn
422                         motors(vL, vR)
423                         script.wait(30)
424                 }
425         } else {
426                 while (encoderL() >= L){
427                         err = Math.abs((encoderL() - encL)) - Math.abs((encoderR() -
                        ↪  encR))
428                         u = err * 2
429                         vL = (60 - u) * sgn
430                         vR = (60 + u) * sgn
431                         motors(vL, vR)
432                         script.wait(30)
433                 }
434         }
435         motors(0)
436 }
437
438 function turnGyro(side, v){
439         //  side: -1 -> left;  1 -> right; 2 - turn back
440         ///
441         v = v || robot.v/2
442         forward(17.5 * 0.5, true)
443         var deltaAngle =
        ↪  gyroAngles[robot.dir]                                                    //
        ↪  current direction
444         robot.dir = (robot.dir + (side < 0 ? 3 : side)) % 4                          //
        ↪  new direction
445         var angle =
        ↪  gyroAngles[robot.dir]
446
447         deltaAngle -= angle
448         angle *= 1000
```

```
449
450         var sgn = sign(deltaAngle) * -1
451         if (abs(deltaAngle) > 180) sgn *= -1
452
453         motors(v * sgn, -v * sgn)
454         var dGyro = 1500
455         while (true){
456                 var yaw = getYaw()
457                 if (angle === 180000){ if (abs(yaw) - dGyro < angle && abs(yaw) +
                    ↪  dGyro > angle) break }
458                 else {if (yaw - dGyro < angle && yaw + dGyro > angle) break }
459                 wait(10)
460         }
461         motors(0)
462         forward(17.5 * -0.5, true)
463 }
464
465 function readSensors(){
466         var s = brick.sensor
467         return [
468                 s('A3').read() > map.cellLength ? 1 : 0,
469                 s('A1').read() > map.cellLength ? 1 : 0,
470                 s('A2').read() > map.cellLength ? 1 : 0,
471         ].join('')
472 }
473
474 function mapping(dir, mapH, mapW){
475         function cellsAround(){
476                 var cells = [robot.cell - map.size, robot.cell + 1, robot.cell +
                    ↪  map.size, robot.cell - 1]
477                 var dirs = [(robot.dir + 3) % 4,
478                                         robot.dir,
479                                         (robot.dir + 1) % 4]
480
481                 return [cells[dirs[0]], cells[dirs[1]], cells[dirs[2]]]
482         }
483
484         function updateMatrix(sensors, cells){
485                 for (var i = 0; i < 3; i++){
486                         if (cells[i] < 0 || cells[i] > map.visited.length-1)
487                                 continue
488
489                         map.matrix[robot.cell][cells[i]] = sensors[i]
490                         map.matrix[cells[i]][robot.cell] = sensors[i]
491
492                         if (map.visited[cells[i]] == 0 && sensors[i] == 1)
493                                 map.visited[cells[i]] = 2
494                         else if (map.visited[cells[i]] == 0)
495                                 map.visited[cells[i]] = sensors[i]
496                 }
497                 minXY(robot.cell)
498         }
499
500         function minXY(cell){
501                 var xy = cell2coord(cell)
502                 if (xy[0] < map.dY) map.dY = xy[0]
503                 if (xy[1] < map.dX) map.dX = xy[1]
504                 map.dCell = coord2cell(map.dY, map.dX)
505         }
506         //
```

```
507         gyroAngles = setsAngles[dir]
508         mapW = mapW || mapH
509 
510         var mapSize = Math.max(mapH, mapW)
511 
512         robot.y = mapSize - 1
513         robot.x = mapSize - 1
514         robot.dir = dir
515 
516         map.size = mapSize * 2 - 1
517         map.cols = map.size, map.rows = map.size
518         robot.cell = coord2cell(robot.y, robot.x)
519 
520         for (var i = 0; i < map.size * map.size; i++){
521                 map.visited.push(0)
522                 map.matrix.push([])
523                 for (var j = 0; j < map.size * map.size; j++)
524                         map.matrix[i].push(-1)
525         }
526 
527 
528 
529 // ***************************** BEGIN ************************
530         map.visited[robot.cell] = 1
531         updateMatrix(readSensors(), cellsAround())
532 
533         if (brick.sensor('A3').read() > map.cellLength)                // left
534                 turnGyro(-1)
535         else
536                 turnGyro(1)
537 
538         updateMatrix(readSensors(), cellsAround())
539 
540         while (true){
541                 var looked = []
542                 // looked but not visited cells
543                 for (var i=0; i < map.visited.length; i++)
544                         if (map.visited[i] == 2) looked.push(i)
545 
546                 if (looked.length == 0) break
547 
548                 var min_len = Infinity, min_path = 0, min_cell = -1
549                 for (var i = 0, p; i < looked.length; i++){
550                         p = bfs(robot.cell, looked[i])
551                         if (p.length < min_len){
552                                 min_len = p.length
553                                 min_path = p
554                                 min_cell = looked[i]
555                         }
556                 }
557                 doCommands(makeCommands(min_path, robot.dir))
558                 map.visited[robot.cell] = 1
559                 robot.updateXY()
560 
561                 updateMatrix(readSensors(), cellsAround())
562                 wait(30)
563         }
564 }
565 
566 //
```

```
//

robot = {
        wheelD: 5.6,
        track: 17.5,
        cpr: 360,
        x: 0, y: 0, cell:0, dir: 1,
        v: 90,
        updateXY: function(){robot.y = Math.floor(robot.cell/map.size); robot.x =
        ↪  robot.cell % map.size},
        updateCell: function(){robot.cell = coord2cell(robot.y, robot.x)}
}

map = {rows: 8, cols: 8, matrix: [], dX: Infinity, dY: Infinity, dCell: 0, visited:
↪  [], size: 8, cellLength: 17.5*4}

robot.cell = coord2cell(robot.y, robot.x)

setsAngles = [[0, 90, 180, -90], [-90, 0, 90, 180], [180, -90, 0, 90],[90, 180, -90,
↪  0]]
gyroAngles = setsAngles[robot.dir]

pi = Math.PI
wait = script.wait
floor = Math.floor
abs = Math.abs

encoderL = brick.encoder('E4').read
encoderR = brick.encoder('E3').read

brick.encoder('E3').reset()
brick.encoder('E4').reset()

pics_raw = script.readAll('input.txt')
artag1 = artag(pics_raw[0].trim().split(' '))
artag2 = artag(pics_raw[1].trim().split(' '))

goalX = artag1, goalY = artag2 % 8
if (Math.max(artag1, artag2) == artag1){
        goalY = artag1 % 8
        goalX = artag2
}
//print ('Y: ',goalY, '   X: ',goalX)

brick.gyroscope().calibrate(2000)
script.wait(2100)

mapping(1, 8)
gotoCell(coord2cell(goalY + map.dY, goalX + map.dX))
brick.display().addLabel('finish',1,1)
brick.display().redraw()
```