2. ВТОРОЙ ЭТАП

Описание этапа

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 138 баллов.

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

Задачи второго этапа

4.1. Задачи

Задача 4.1.1. Запуск всенаправленной тележки (5 баллов)

Робот, моторы которого расположены под углом в 120° друг к другу, движется в некотором направлении, пока на моторы подается мощность. Каждый мотор работает некоторое время: t_1, t_2, t_3, соответственно. На валах моторов закреплены омниколёса (`https://en.wikipedia.org/wiki/Omni_wheel`). С некоторой кинематической моделью робота можно познакомиться по ссылке: `https://bharat-robotics.github.io/blog/kinematic-analysis-of-holonomic-robot/`.

Необходимо определить новые координаты центра робота, если в момент начала движения он находился в точке $(0, 0)$ и один из двигателей находился на оси Y, в направлении положительной части (см. рис. 4.1). Расположение моторов является постоянным и соответствует рисунку 4.1.

Считать, что мощность на все моторы подаётся одновременно и достигается мгновенно. Также считать что происходит движение без поворотов, иначе говоря робот двигается только прямолинейно в любом направлении. Колёса вращаются без проскальзывания.

Данные в тестах подобраны таким образом, что нет вариата движения, когда робот движется вокруг какой-то точки.

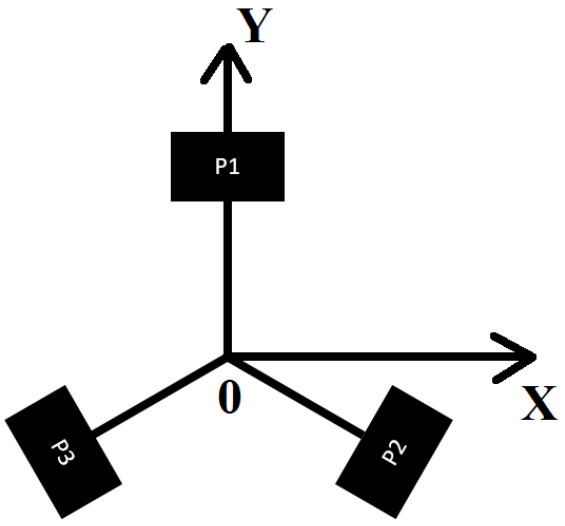


Рис. 4.1: Расположение моторов робота относительно центра глобальной системы отсчета в начальный момент времени

Формат входных данных

Одна строчка, состоящая из 7ми чисел: d, w_1, t_1, w_2, t_2, w_3, t_3, разделёнными пробелами, где

- d — диаметр колёс в мм ($30 \leqslant d \leqslant 100$);
- w_1, w_2, w_3 — скорости вращения моторов в рад/с ($-2 \leqslant w_1, w_2, w_3 \leqslant 2$);
- t_1, t_2, t_3 — время работы каждого мотора в с ($10 \leqslant t_1, t_2, t_3 \leqslant 1000$).

Диаметр колёс и время движения — целые числа, скорости вращения — вещественные.

Комментарии

Дополнительные наборы входных данных доступны по ссылке `http://bit.ly/2RdizA3`.

Формат выходных данных

Одна строка, содержащая два целых числа через пробел — координаты центра робота в мм, где он закончил своё движение. Допускается погрешность в 1 мм по каждой из координат.

Примеры

Пример №1

Стандартный ввод
`35 1 15 0 0 -1 15`
Стандартный вывод
`196.875 -113.666`

Пример №2

Стандартный ввод
`40 1.4 100 -1.4 100 1.4 100`
Стандартный вывод
`2800 2424.87`

Решение

Рассмотрим схему всенаправленной тележки (рис. 4.2).

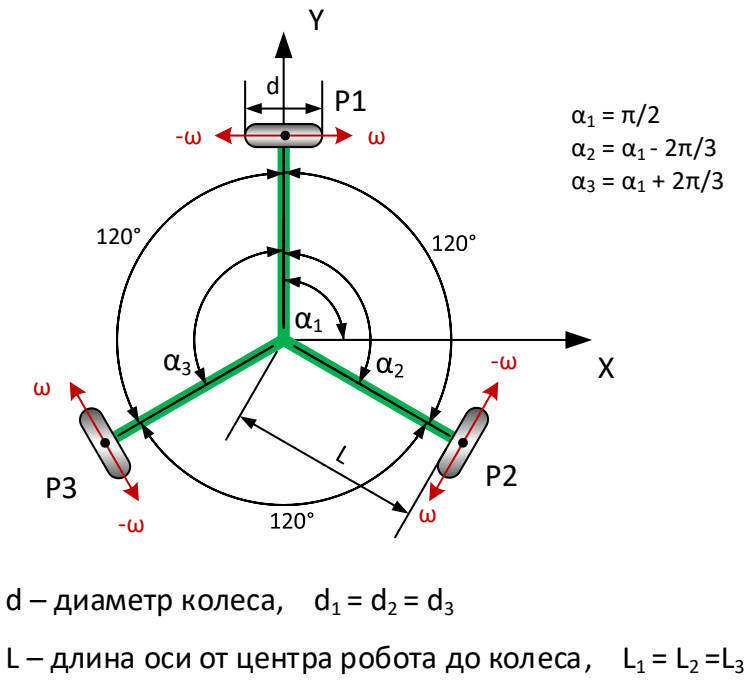


Рис. 4.2: Схема всенаправленной тележки

У каждого мотора может быть 3 основных состояния: 'вперед', 'назад', 'выключен'.

Определим направления вращения моторов: пусть при угловой скорости $\omega_i < 0$ колеса вращаются "вперед"или по часовой стрелке (если смотреть со стороны колес), если $\omega_i > 0$, то колеса вращаются "назад"или против часовой стрелки.

По условиям задачи движение происходит без вращения робота, т.е. он двигается только прямолинейно в любом направлении. Рассмотрим комбинации угловых скоростей (ω), при которых всенаправленная тележка едет только прямо (таблица 4.1), при этом угловые скорости моторов (ω) должны быть или равны по модулю или у одного из моторов $\omega = 0$:

P1	↑	↑	↑	↑	0	0	↓	↓	↓	↓
P2	↓	↓	0	↓	↓	↑	↑	0	↑	↑
P3	↑	0	↓	↓	↑	↓	↑	↑	0	↓

где:
↑ - включение мотора 'вперед',
↓ - включение мотора 'назад',
0 - мотор выключен

Таблица 4.1: Варианты включения моторов для прямолинейного движения всенаправленной тележки

Исходя из этих условий, мы можем использовать упрощенную кинематическую модель, в которой получаем следующие уравнения прямой кинематики (формулы

4.1) и (4.2).

$$x = x_0 + \pi D \left(\omega_1 \cdot sin(\alpha) + \omega_2 \cdot sin\left(\alpha - \frac{2\pi}{3}\right) + \omega_3 \cdot sin\left(\alpha + \frac{2\pi}{3}\right)\right) \cdot t \tag{4.1}$$

$$y = y_0 + \pi D \left(\omega_1 \cdot cos(\alpha) + \omega_2 \cdot cos\left(\alpha - \frac{2\pi}{3}\right) + \omega_3 \cdot cos\left(\alpha + \frac{2\pi}{3}\right)\right) \cdot t \tag{4.2}$$

Поскольку в задаче сказано, что ось первого колеса совпадает с осью Y, то в нашем случае $\alpha = \pi/2$, тогда формулы принимают вид (формулы 4.3 и 4.4):

$$x = x_0 + \pi D \left(\omega_1 \cdot t_1 \cdot sin\left(\frac{\pi}{2}\right) + \omega_2 \cdot t_2 \cdot sin\left(\frac{\pi}{2} - \frac{2\pi}{3}\right) + \omega_3 \cdot t_3 \cdot sin\left(\frac{\pi}{2} + \frac{2\pi}{3}\right)\right) \tag{4.3}$$

$$y = y_0 + \pi D \left(\omega_1 \cdot t_1 \cdot cos\left(\frac{\pi}{2}\right) + \omega_2 \cdot t_2 \cdot cos\left(\frac{\pi}{2} - \frac{2\pi}{3}\right) + \omega_3 \cdot t_3 \cdot cos\left(\frac{\pi}{2} + \frac{2\pi}{3}\right)\right) \tag{4.4}$$

где x_0 и y_0 – координаты робота на плоскости в начальный момент времени

После упрощения формула принимают вид:

$$x = x_0 + \pi D \left(\omega_1 \cdot t_1 + \omega_2 \cdot t_2 \cdot cos\left(-\frac{2\pi}{3}\right) + \omega_3 \cdot t_3 \cdot cos\left(\frac{2\pi}{3}\right)\right) \tag{4.5}$$

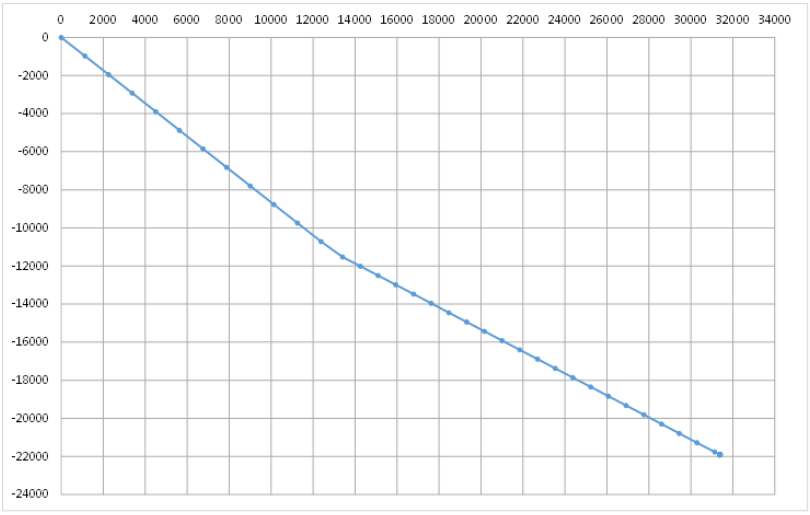
$$y = y_0 + \pi D \left(\omega_1 \cdot t_1 + \omega_2 \cdot t_2 \cdot sin\left(-\frac{2\pi}{3}\right) + \omega_3 \cdot t_3 \cdot sin\left(\frac{2\pi}{3}\right)\right) \tag{4.6}$$

В какой-то момент времени один из моторов останавливается. Следовательно, необходимо выполнить две последовательные итерации вычисления координат:

- в первой итерации вращаются все три колеса
- во второй итерации - только два колеса (т.к. у третьего колеса $\omega = 0$)

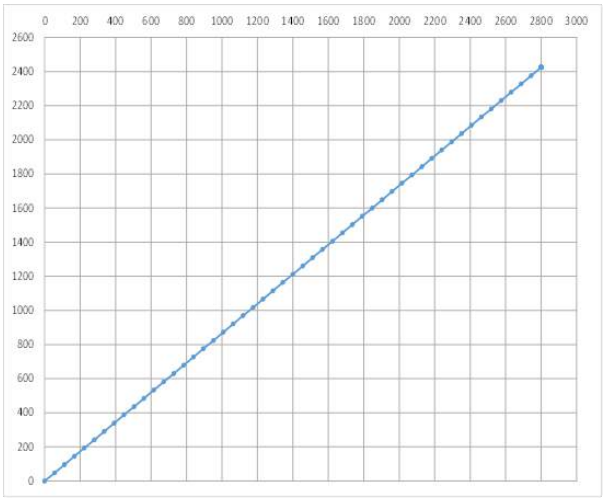
После остановки всех моторов мы получаем конечные координаты робота X и Y, которые и выводим в ответ.

Примеры графиков движений по наборам данных представлены на рис.4.3 и 4.4.



d	w_1	w_2	w_3	t_1	t_2	t_3	⟹	x	y
73	1.542	1.542	-1.542	666	233	666		31391.8	-21909.8

Рис. 4.3: График движения всенаправленной тележки (пример 1)



d	w_1	w_2	w_3	t_1	t_2	t_3	⟹	x	y
40	1.4	-1.4	1.4	100	100	100		2800.0	2424.9

Рис. 4.4: График движения всенаправленной тележки (пример 2)

Пример программы-решения

Ниже представлено решение на языке C++

```
#include <iostream>
#include <cmath>

#define PI 3.1415926

class Vector
{
```

```
8  public:
9          Vector(double endX, double endY)
10         {
11                 this->x = endX;
12                 this->y = endY;
13         }
14         void setX(double endX) { this->x = endX; }
15         void setY(double endY) { this->y = endY; }
16         double getX() { return x; }
17         double getY() { return y; }
18         void add(Vector vec)
19         {
20                 x += vec.getX();
21                 y += vec.getY();
22         }
23         void subtract(Vector vec)
24         {
25                 x -= vec.getX();
26                 y -= vec.getY();
27         }
28         void mult(double num)
29         {
30                 x *= num;
31                 y *= num;
32         }
33         Vector operator+(Vector vec)
34         {
35                 Vector newVec(*this);
36                 newVec.add(vec);
37                 return newVec;
38         }
39         Vector operator+=(Vector vec)
40         {
41                 add(vec);
42                 return *this;
43         }
44         Vector operator-(Vector vec)
45         {
46                 Vector newVec(*this);
47                 newVec.subtract(vec);
48                 return newVec;
49         }
50         Vector operator-=(Vector vec)
51         {
52                 subtract(vec);
53                 return *this;
54         }
55         Vector operator*(double num)
56         {
57                 Vector newVec(*this);
58                 newVec.mult(num);
59                 return newVec;
60         }
61         Vector operator*=(double num)
62         {
63                 mult(num);
64                 return *this;
65         }
66         bool operator==(int zero)
67         {
```

```
68                  return zero == 0 && x == 0 && y == 0;
69          }
70
71  private:
72          double x;
73          double y;
74  };
75
76  struct Motion
77  {
78  public:
79          Motion() {}
80          Motion(double m1w, double m2w, double m3w, double time)
81          {
82                  motor1w = m1w;
83                  motor2w = m2w;
84                  motor3w = m3w;
85                  this->time = time;
86          }
87
88          double
89                  motor1w,
90                  motor2w,
91                  motor3w;
92          double time;
93          double radius;
94
95          int getType()
96          {
97                  if (time == 0) return 0;
98                  if (motor1w == motor2w && motor2w == motor3w) return 0;
99                  if (motor1w*motor2w*motor3w == 0) return 1;
100                 else return 2;
101         }
102         Vector getMotionVector()
103         {
104                 int type = getType();
105                 switch (type)
106                 {
107                 case 0: return Vector(0, 0);
108                 case 1:
109                         if (motor1w == 0)
110                         {
111                                 double speed = motor2w * radius;
112                                 Vector sVec(0, -speed * cos(PI / 6));
113                                 return sVec;
114                         }
115                         if (motor2w == 0)
116                         {
117                                 double speed = motor1w * radius;
118                                 Vector sVec(speed * pow(cos(PI / 6), 2), -speed
119                                         * sin(PI / 6) * cos(PI / 6));
120                                 return sVec;
121                         }
122                         if (motor3w == 0)
123                         {
124                                 double speed = motor1w * radius;
125                                 Vector sVec(speed * pow(cos(PI / 6), 2), speed
126                                         * sin(PI / 6) * cos(PI / 6));
127                                 return sVec;
```

```
128                         }
129                 case 2:
130                         if (motor1w == motor2w)
131                         {
132                                 double speed3 = motor3w * radius;
133                                 return Vector(motor1w*radius, speed3*sin(PI / 3));
134                         }
135                         if (motor1w == motor3w)
136                         {
137                                 double speed2 = motor2w * radius;
138                                 return Vector(motor1w*radius, -speed2 * sin(PI / 3));
139                         }
140                         if (motor2w == motor3w)
141                         {
142                                 double speed1 = motor1w * radius;
143                                 return Vector(speed1, 0);
144                         }
145                 }
146         }
147 };
148
149 using namespace std;
150
151 int main()
152 {
153         double d, w1, t1, w2, t2, w3, t3;
154         cin >> d >> w1 >> t1 >> w2 >> t2 >> w3 >> t3;
155
156         //split motion into simple motions by timespans
157         Motion m1, m2, m3;
158         m1.radius = m2.radius = m3.radius = d / 2;
159         if (t1 >= t2 && t2 >= t3)
160         {
161                 m1.motor1w = w1; m1.motor2w = w2; m1.motor3w = w3;
162                 m1.time = t3;
163                 m2.motor1w = w1; m2.motor2w = w2; m2.motor3w = 0;
164                 m2.time = t2 - t3;
165                 m3.motor1w = w1; m3.motor2w = 0; m3.motor3w = 0;
166                 m3.time = t1 - t2;
167         }
168         else if (t1 >= t3 && t3 >= t2)
169         {
170                 m1.motor1w = w1; m1.motor2w = w2; m1.motor3w = w3;
171                 m1.time = t2;
172                 m2.motor1w = w1; m2.motor2w = 0; m2.motor3w = w3;
173                 m2.time = t3 - t2;
174                 m3.motor1w = w1; m3.motor2w = 0; m3.motor3w = 0;
175                 m3.time = t1 - t3;
176         }
177         else if (t2 >= t1 && t1 >= t3)
178         {
179                 m1.motor1w = w1; m1.motor2w = w2; m1.motor3w = w3;
180                 m1.time = t3;
181                 m2.motor1w = w1; m2.motor2w = w2; m2.motor3w = 0;
182                 m2.time = t1 - t3;
183                 m3.motor1w = 0; m3.motor2w = w2; m3.motor3w = 0;
184                 m3.time = t2 - t1;
185         }
186         else if (t2 >= t3 && t3 >= t1)
187         {
```

```
188                 m1.motor1w = w1; m1.motor2w = w2; m1.motor3w = w3;
189                 m1.time = t1;
190                 m2.motor1w = 0; m2.motor2w = w2; m2.motor3w = w3;
191                 m2.time = t3 - t1;
192                 m3.motor1w = 0; m3.motor2w = w2; m3.motor3w = 0;
193                 m3.time = t2 - t3;
194         }
195         else if (t3 >= t1 && t1 >= t2)
196         {
197                 m1.motor1w = w1; m1.motor2w = w2; m1.motor3w = w3;
198                 m1.time = t2;
199                 m2.motor1w = w1; m2.motor2w = 0; m2.motor3w = w3;
200                 m2.time = t1 - t2;
201                 m3.motor1w = 0; m3.motor2w = 0; m3.motor3w = w3;
202                 m3.time = t3 - t1;
203         }
204         else if (t3 >= t2 && t2 >= t1)
205         {
206                 m1.motor1w = w1; m1.motor2w = w2; m1.motor3w = w3;
207                 m1.time = t1;
208                 m2.motor1w = 0; m2.motor2w = w2; m2.motor3w = w3;
209                 m2.time = t2 - t1;
210                 m3.motor1w = 0; m3.motor2w = 0; m3.motor3w = w3;
211                 m3.time = t3 - t2;
212         }
213
214         //starting motion
215         Vector result(0, 0);
216         result += m1.getMotionVector() * m1.time;
217         result += m2.getMotionVector() * m2.time;
218         result += m3.getMotionVector() * m3.time;
219
220         cout << result.getX() << ' ' << result.getY() << endl;
221 }
```

Задача 4.1.2. Управление всенаправленной тележкой (10 баллов)

Робот, моторы которого расположены под углом в 120° друг к другу, движется в заданном направлении некоторое время t. На валах моторов закреплены омниколёса (`https://en.wikipedia.org/wiki/Omni_wheel`). С некоторой кинематической моделью робота можно познакомиться по ссылке: `https://bharat-robotics.github.io/blog/kinematic-analysis-of-holonomic-robot/`.

Необходимо определить новые координаты центра робота, если в момент начала движения он находился в точке $(0, 0)$ и один из двигателей находился на оси Y, в направлении положительной части (см. рис. 4.5). Расположение моторов является постоянным и соответствует рисунку 4.5.

Считать, что мощность достигается мгновенно и может подаваться на все моторы одновременно. Колёса вращаются без проскальзывания. В случае когда на моторы ничего не подаётся, их скорость равна 0.

Формат входных данных

Первая строчка содержит четыре целых числа через пробел – d, p, t, N, где:

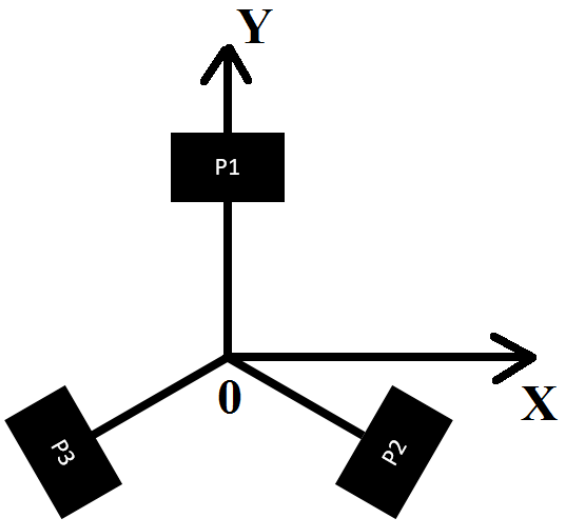


Рис. 4.5: Расположение моторов робота относительно центра глобальной системы отсчета в начальный момент времени

- d — диаметр колёс в мм ($30 \leqslant d \leqslant 100$);
- p — длина оси (осевой балки) от центра робота до колеса в мм ($50 \leqslant p \leqslant 125$);
- t — общее время работы моторов в с ($10 \leqslant t \leqslant 1000$);
- N — количество измерений ($1 \leqslant N \leqslant 1000$).

Далее идут 3 строки — для первого, второго и третьего моторов, соотвественно.

В каждой строке находится N вещественных чисел через пробел — подаваемая на мотор скорость w_i, через равные промежутки времени. (-2 рад/с $\leqslant w_i \leqslant 2$ рад/с)

Формат выходных данных

Одна строка, содержащая два целых числа через пробел — координаты центра робота в мм, где он закончил своё движение. Допускается погрешность в 1 мм по каждой из координат.

Примеры

Пример №1

Стандартный ввод

`30 50 12 4`
`0 0 2 2`
`-2 -2 0 0`
`2 2 2 2`

Стандартный вывод
`77.486 167.373`

Пример №2

Стандартный ввод
35 55 16 8 -0.8 -0.8 -0.8 0.6 0.6 0.6 0 0 -0.8 -0.8 -0.8 0 0 0 0 0 0.8 0.8 0.8 0.6 0.6 0.6 0 0

Стандартный вывод
1.504 130.544

Решение

В данной задаче модель всенаправленной тележки аналогична модели в предыдущей задаче (рис. 4.6) и по условиям задачи тележка движется прямолинейно, т.е. мы можем использовать уравнения кинематики из прошлой задачи.

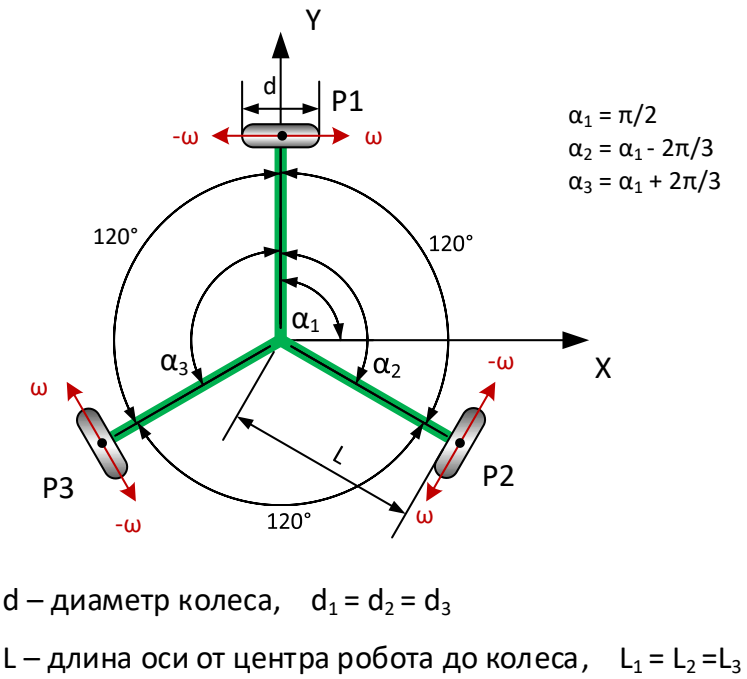


Рис. 4.6: Схема всенаправленной тележки

Рассмотрим отличия от предыдущей задачи:

- Время вращения всех моторов постоянное и одинаковое, t с.
- У моторов (у всех одновременно) происходит изменение угловых скоростей, N раз за все время движения t

Время, используемое в этой формуле, вычисляется делением всего временного интервала, в течение которого двигается робот на количество измерений (формула 4.7).

$$dt = \frac{t}{N} \tag{4.7}$$

Таким образом, чтобы получить итоговые координаты центра тележки, нужно

итеративно (через время dt) производить вычисление новых координат, подставляя в каждой итерации очередное значение мощности для каждого мотора.

Пример программы-решения

Ниже представлено решение на языке C++

```
#include <iostream>
#include <cmath>
#include <algorithm>
#include <vector>
#include <iomanip>

#define PI 3.1415926

using namespace std;

class Point
{
public:
        Point(double x, double y) { _x = x; _y = y; }
        Point() { _x = _y = 0; }
        double getX() { return _x; }
        double getY() { return _y; }
        void add(Point p) { _x += p._x; _y += p._y; }
        void subtract(Point p) { _x -= p._x; _y -= p._y; }
        void multiply(double num) { _x *= num; _y *= num; }
        Point operator+(Point p)
        {
                Point newP(*this);
                newP.add(p);
                return newP;
        }
        Point operator-(Point p)
        {
                Point newP(*this);
                newP.subtract(p);
                return newP;
        }
        Point operator*(double num)
        {
                Point newP(*this);
                newP.multiply(num);
                return newP;
        }
        Point operator+=(Point p)
        {
                add(p);
                return *this;
        }
        Point operator-=(Point p)
        {
                subtract(p);
                return *this;
        }
        Point operator*=(double num)
        {
                multiply(num);
                return *this;
```

```
53          }
54          bool operator==(Point p) { return abs(_x - p._x) < 1E-2 &&
55                  abs(_y - p._y) < 1E-2; }
56  private:
57          double _x, _y;
58  };
59
60  class Vector
61  {
62  public:
63          Vector(Point start, Point end)
64          {
65                  _start = start;
66                  _end = end;
67          }
68          Point getStart() { return _start; }
69          Point getEnd() { return _end; }
70          void add(Vector vec)
71          {
72                  double
73                          offsetX = vec._end.getX() - vec._start.getX(),
74                          offsetY = vec._end.getY() - vec._start.getY();
75                  _end += Point(offsetX, offsetY);
76          }
77          void subtract(Vector vec)
78          {
79                  double
80                          offsetX = vec._end.getX() - vec._start.getX(),
81                          offsetY = vec._end.getY() - vec._start.getY();
82                  _end -= Point(offsetX, offsetY);
83          }
84          void multiply(double num)
85          {
86                  Point unit = _end - _start;
87                  unit *= num;
88                  _end = _start + unit;
89          }
90          Vector operator+(Vector vec)
91          {
92                  Vector newVec(*this);
93                  newVec.add(vec);
94                  return newVec;
95          }
96          Vector operator+=(Vector vec)
97          {
98                  add(vec);
99                  return *this;
100         }
101         Vector operator-(Vector vec)
102         {
103                 Vector newVec(*this);
104                 newVec.subtract(vec);
105                 return newVec;
106         }
107         Vector operator-=(Vector vec)
108         {
109                 subtract(vec);
110                 return *this;
111         }
112         Vector operator*(double num)
```

```
113         {
114                 Vector newVec(*this);
115                 newVec.multiply(num);
116                 return newVec;
117         }
118         Vector operator*=(double num)
119         {
120                 multiply(num);
121                 return *this;
122         }
123         bool operator==(int zero)
124         {
125                 return zero == 0 && _end - _start == Point(0, 0);
126         }
127         operator Point()
128         {
129                 return _end - _start;
130         }
131         double lenght()
132         {
133                 Point p = Point(*this);
134                 return sqrt(pow(p.getX(), 2) + pow(p.getY(), 2));
135         }
136
137 private:
138         Point _start;
139         Point _end;
140 };
141
142 struct Motion
143 {
144 public:
145         Motion() {}
146         Motion(double m1w, double m2w, double m3w, double time)
147         {
148                 motor1w = m1w;
149                 motor2w = m2w;
150                 motor3w = m3w;
151                 this->time = time;
152         }
153
154         double
155                 motor1w,
156                 motor2w,
157                 motor3w;
158         double time;
159         double radius;
160 };
161
162 class Robot
163 {
164 public:
165         Robot(double originDistance, double wheelRadius)
166         {
167                 _wheelRadius = wheelRadius;
168                 _originDistance = originDistance;
169                 _orientation = 0;
170                 _origin = Point(0, 0);
171                 _motor1 = Point(originDistance, 0);
172                 _motor2 = Point(originDistance * cos(PI / 6), -originDistance
```

```
173                     * sin(PI / 6));
174             _motor3 = Point(-originDistance * cos(PI / 6), -originDistance
175             * sin(PI / 6));
176     }
177     void startMotion(Motion motion)
178     {
179             if (_origin == Point(0, 0))
180             {
181                     double tOrientation = _orientation;
182                     if (motion.motor1w == -motion.motor2w)
183                     {
184                             rotate(2. / 3 * PI - tOrientation);
185                             Vector projX(Point(0, 0), Point(motion.motor3w
186                                     * motion.radius, 0));
187                             Vector projY(Point(0, 0), Point(0, -motion.motor1w
188                                     * motion.radius*sin(PI / 3)));
189                             Vector res = (projX + projY)*motion.time;
190                             _motor1 += (Point)res;
191                             _motor2 += (Point)res;
192                             _motor3 += (Point)res;
193                             _origin += (Point)res;
194                             rotate(-2. / 3 * PI + tOrientation);
195                             return;
196                     }
197                     if (motion.motor1w == -motion.motor3w)
198                     {
199                             rotate(-2. / 3 * PI - tOrientation);
200                             Vector projX(Point(0, 0), Point(motion.motor2w
201                                     * motion.radius, 0));
202                             Vector projY(Point(0, 0), Point(0, motion.motor1w
203                                     * motion.radius * sin(PI / 3)));
204                             Vector res = (projX + projY)*motion.time;
205                             _motor1 += (Point)res;
206                             _motor2 += (Point)res;
207                             _motor3 += (Point)res;
208                             _origin += (Point)res;
209                             rotate(2. / 3 * PI + tOrientation);
210                             return;
211                     }
212                     if (motion.motor2w == -motion.motor3w)
213                     {
214                             Vector projX(Point(0, 0), Point(motion.motor1w
215                                     * motion.radius, 0));
216                             Vector projY(Point(0, 0), Point(0, -motion.motor2w
217                                     * motion.radius*sin(PI / 3)));
218                             Vector res = (projX + projY)*motion.time;
219                             _motor1 += (Point)res;
220                             _motor2 += (Point)res;
221                             _motor3 += (Point)res;
222                             _origin += (Point)res;
223                             return;
224                     }
225                     if (motion.motor1w == motion.motor2w)
226                     {
227                             rotate(-2. / 3 * PI - tOrientation);
228                             Vector proj(Point(0, 0), Point(motion.motor2w
229                                     * motion.radius - motion.motor1w*motion.radius
230                                     * cos(PI / 3), motion.motor1w*motion.radius
231                                     * sin(PI / 3)));
232                             if ((proj - Vector(Point(0, 0), Point(-motion.motor3w
```

```
233                             * motion.radius*cos(PI / 6), -motion.motor3w
234                             * motion.radius*sin(PI / 6))) == 0))
235                     {
236                             _motor1 += (Point)proj;
237                             _motor2 += (Point)proj;
238                             _motor3 += (Point)proj;
239                             _origin += (Point)proj;
240                             rotate(2. / 3 * PI + tOrientation);
241                             return;
242                     }
243                     Vector speed1(Point(0, 0), Point(-motion.motor3w
244                             * motion.radius*cos(PI / 6), -motion.motor3w
245                             * motion.radius*sin(PI / 6)));
246                     double proj1 = proj.lenght();
247                     if (motion.motor2w < 0) proj1 *= -1;
248                     double proj2 = speed1.lenght();
249                     if (motion.motor3w < 0) proj2 *= -1;
250                     double d = (proj2*_originDistance - proj1
251                             * _originDistance) / (proj1 + proj2);
252                     Point center = _motor3 * -(d / _originDistance);
253                     double angle = proj1 * motion.time
254                             / (_originDistance - d);
255                     _motor1 -= center;
256                     _motor2 -= center;
257                     _motor3 -= center;
258                     _origin -= center;
259                     rotate(angle);
260                     _motor1 += center;
261                     _motor2 += center;
262                     _motor3 += center;
263                     _origin += center;
264                     rotate(2. / 3 * PI + tOrientation);
265                     return;
266             }
267             if (motion.motor2w == motion.motor3w)
268             {
269                     rotate(2. / 3 * PI - tOrientation);
270                     Vector proj(Point(0, 0), Point(motion.motor3w
271                             * motion.radius - motion.motor2w*motion.radius
272                             * cos(PI / 3), motion.motor2w*motion.radius
273                             * sin(PI/3)));
274                     if ((proj - Vector(Point(0, 0), Point(-motion.motor1w
275                             * motion.radius*cos(PI/6), -motion.motor1w
276                             * motion.radius*sin(PI/6))) == 0))
277                     {
278                             _motor1 += (Point)proj;
279                             _motor2 += (Point)proj;
280                             _motor3 += (Point)proj;
281                             _origin += (Point)proj;
282                             rotate(-2. / 3 * PI + tOrientation);
283                             return;
284                     }
285                     Vector speed1(Point(0, 0), Point(-motion.motor1w
286                             * motion.radius*cos(PI / 6), -motion.motor1w
287                             * motion.radius*sin(PI / 6)));
288                     double proj1 = proj.lenght();
289                     if (motion.motor2w < 0) proj1 *= -1;
290                     double proj2 = speed1.lenght();
291                     if (motion.motor1w < 0) proj2 *= -1;
292                     double d = (proj2*_originDistance - proj1
```

```
293                                     * _originDistance) / (proj1 + proj2);
294                             Point center = _motor1* -(d / _originDistance);
295                             double angle = proj1 * motion.time
296                                     / (_originDistance - d);
297                             _motor1 -= center;
298                             _motor2 -= center;
299                             _motor3 -= center;
300                             _origin -= center;
301                             rotate(angle);
302                             _motor1 += center;
303                             _motor2 += center;
304                             _motor3 += center;
305                             _origin += center;
306                             rotate(-2. / 3 * PI + tOrientation);
307                             return;
308                     }
309                     if (motion.motor3w == motion.motor1w)
310                     {
311                             Vector proj(Point(0, 0), Point(motion.motor1w
312                                     * motion.radius - motion.motor3w*motion.radius
313                                     * cos(PI / 3), motion.motor3w*motion.radius
314                                     * sin(PI / 3)));
315                             if ((proj - Vector(Point(0, 0), Point(-motion.motor2w
316                                     * motion.radius*cos(PI / 6), -motion.motor2w
317                                     * motion.radius*sin(PI / 6))) == 0))
318                             {
319                                     _motor1 += (Point)proj;
320                                     _motor2 += (Point)proj;
321                                     _motor3 += (Point)proj;
322                                     _origin += (Point)proj;
323                                     return;
324                             }
325                             Vector speed1(Point(0, 0), Point(-motion.motor2w
326                                     * motion.radius*cos(PI / 6), -motion.motor2w
327                                     * motion.radius*sin(PI / 6)));
328                             double proj1 = proj.lenght();
329                             if (motion.motor3w < 0) proj1 *= -1;
330                             double proj2 = speed1.lenght();
331                             if (motion.motor2w < 0) proj2 *= -1;
332                             double d = (proj2*_originDistance - proj1
333                                     * _originDistance) / (proj1 + proj2);
334                             Point center = _motor2 * -(d / _originDistance);
335                             double angle = proj1 * motion.time
336                                     / (_originDistance - d);
337                             _motor1 -= center;
338                             _motor2 -= center;
339                             _motor3 -= center;
340                             _origin -= center;
341                             rotate(angle);
342                             _motor1 += center;
343                             _motor2 += center;
344                             _motor3 += center;
345                             _origin += center;
346                             return;
347                     }
348             }
349             Point tOrigin = _origin;
350             _motor1 -= _origin;
351             _motor2 -= _origin;
352             _motor3 -= _origin;
```

```
353                 _origin -= tOrigin;
354                 startMotion(motion);
355                 _motor1 += tOrigin;
356                 _motor2 += tOrigin;
357                 _motor3 += tOrigin;
358                 _origin += tOrigin;
359         }
360         void rotate(double angle)
361         {
362                 double
363                         cosa = cos(angle),
364                         sina = sin(angle);
365 
366                 _motor1 = Point(_motor1.getX()*cosa + _motor1.getY()*sina,
367                         -_motor1.getX()*sina + _motor1.getY()*cosa);
368                 _motor2 = Point(_motor2.getX()*cosa + _motor2.getY()*sina,
369                         -_motor2.getX()*sina + _motor2.getY()*cosa);
370                 _motor3 = Point(_motor3.getX()*cosa + _motor3.getY()*sina,
371                         -_motor3.getX()*sina + _motor3.getY()*cosa);
372                 _origin = Point(_origin.getX()*cosa + _origin.getY()*sina,
373                         -_origin.getX()*sina + _origin.getY()*cosa);
374 
375                 _orientation += angle;
376         }
377         Point getOrigin() { return _origin; }
378 
379 private:
380         Point _origin;
381         Point
382                 _motor1,
383                 _motor2,
384                 _motor3;
385         double
386                 _orientation,
387                 _wheelRadius,
388                 _originDistance;
389 };
390 
391 
392 int main()
393 {
394         double d, r, t, n;
395         cin >> d >> r >> t >> n;
396 
397         vector<double> w1(n), w2(n), w3(n);
398         for (int i = 0; i < n; i++) cin >> w1[i];
399         for (int i = 0; i < n; i++) cin >> w2[i];
400         for (int i = 0; i < n; i++) cin >> w3[i];
401 
402         Motion m;
403         m.radius = m.radius = m.radius = d / 2;
404         m.time = t / n;
405 
406         //starting motion
407         Robot robot(r, m.radius);
408         for (int i = 0; i < n; i++)
409         {
410                 m.motor1w = w1[i];
411                 m.motor2w = w2[i];
412                 m.motor3w = w3[i];
```

```
413                 robot.startMotion(m);
414         }
415
416         cout << fixed << setprecision(3) << robot.getOrigin().getX() << ' '
417                  << robot.getOrigin().getY() << endl;
418 }
```

Задача 4.1.3. Определение размера препятствия (15 баллов)

Робот, собранный по дифференциальной схеме, передвигается по полигону. Он оснащён приёмником и дальномером. Приёмник позволяет измерять расстояния до установленных на поле и находящихся в прямой видимости маяков. Дальномер направлен влево по ходу движения робота и позволяет получать информацию о находящихся препятствиях в данном направлении. На полигоне находятся препятствие и K маяков, координаты которых передаются через входной файл. Гарантируется, что маяки не мешают роботу перемещаться и не попадают в поле зрения дальномера. Пример полигона представлен на рис. 4.7.

Необходимо найти площадь препятствия, расположенного на полигоне. Известно что, приёмник возвращает значения расстояний до препятствия в мм, считая от оси вращения робота, находящейся в центре между колёсами. Дальномер расположен в том же месте, что и приёмник.

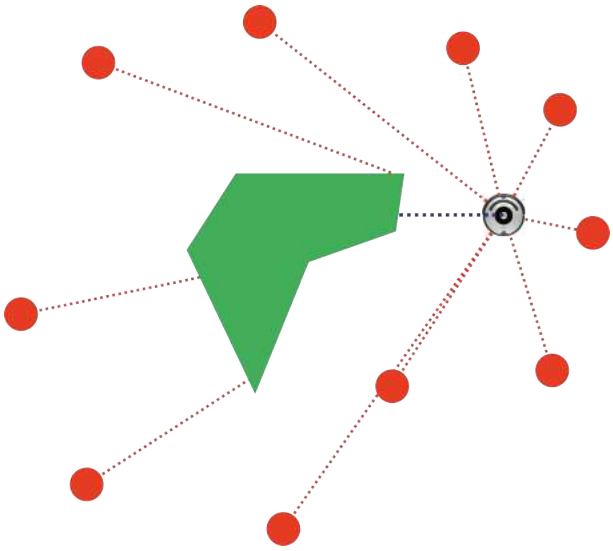


Рис. 4.7: Пример соревновательного полигона

Формат входных данных

Первая строка содержит 3 целых числа: K и N и dT:

- K — количество маяков ($10 \leqslant K \leqslant 100$);
- N — количество замеров ($10 \leqslant N \leqslant 10\ 000$);
- dT — пауза между измерениями в мс ($0 \leqslant dT \leqslant 10\ 000$).

Далее идёт K строк. Каждая строка имеет следующую структуру: i, x_i, y_i, в которой все числа вещественные и разделены пробелами, где:

- i — номер маяка по порядку ($0 \leqslant i \leqslant 10\ 000$);

- x_i — координата x i-того маяка ($-10\ 000 \leqslant x_i \leqslant 10\ 000$);
- y_i — координата y i-того маяка ($-10\ 000 \leqslant y_i \leqslant 10\ 000$).

Далее идёт N строк. Каждая строка содержит:

- d — целое число, показание дальномера в мм ($0 \leqslant d \leqslant 1\ 000$), в случае если предмет находится вне поля видимости дальномера, то его значение будет равно 1 000;
- $Value_1\ Value_2\ \ldots\ Value_k$ — вещественные числа, показания, получаемые приёмником с каждого маяка. В случае, если маяк находится вне зоны видимости, то соответствующее значение будет равно -1.

Все числа указаны через пробел.

Формат выходных данных

Одна строка, содержащая одно целове число – площадь препятствия в мм^2. Допускается погрешность в ± 1 м^2.

Примеры

Примеры входных данных и ответов к ним можно найти по сслыке `http://bit.ly/2Re5l9N`.

Решение

Для определения границ препятствия сначала определим координаты робота в момент замеров. Для этого мы найдем точки пересечения двух окружностей, образованных от центров маяков, которые "видят"робота в данный момент времени. Схема представлена на рис.4.8, где:

- M_1 - маяк N, установленный в координатах $(a_1; b_1)$
- M_2 - маяк K, установленный в координатах $(a_2; b_2)$
- r_1 - радиус первой окружности (расстояние от маяка N до центра робота)
- r_2 - радиус второй окружности (расстояние от маяка K до центра робота)

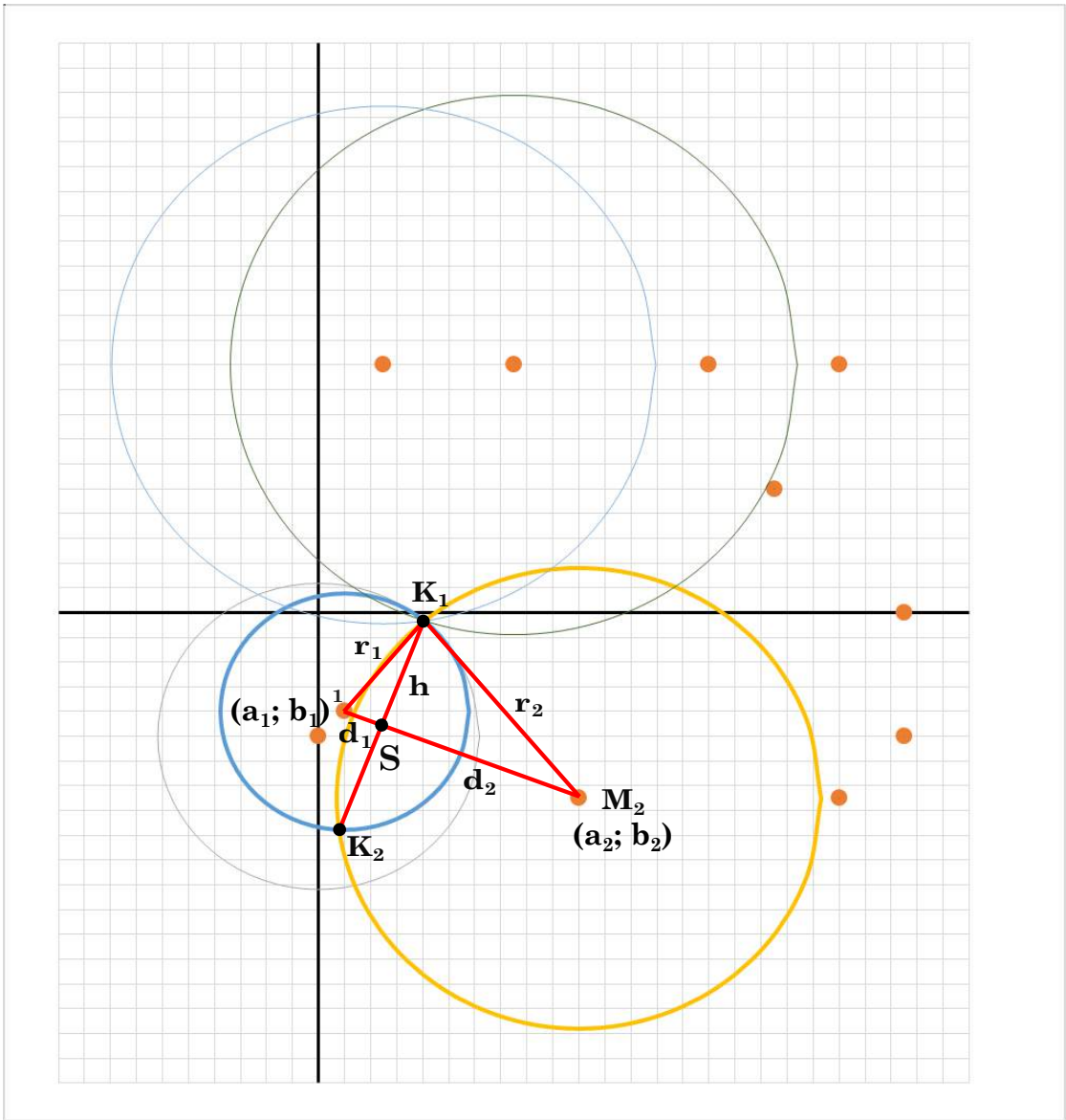


Рис. 4.8:

Сначала найдем расстояние между центрами маяков: d $(M_1;M_2)$ (можно включить проверку отсутствие решений: $d = |M_1 - M_2|$, при $d > r_1 + r_2$ (круги не пересекаются) или $d < |r_2 - r_1|$ (одна окружность находится внутри другой)). На схеме (рис.4.8) видно, что r_1 и r_2 - катеты $\triangle M_1M_2K_1$, а искомая сторона M_1M_2 - гипотенуза. Исходя из теоремы Пифагора, найдем гипотенузу по формуле (4.8).

$$d = \sqrt{(a_2 - a_1)^2 + (b_2 - b_1)^2} \tag{4.8}$$

Далее нам надо найти r_2, $M_1S = d_1$; $M_2S = d_2$.

$$\begin{cases} d_1 + d_2 = d \\ h = \sqrt{{r_1}^2 - {d_1}^2} = \sqrt{{r_2}^2 - {d_2}^2} \end{cases} \Rightarrow \begin{cases} d_1 + d_2 = d \\ {r_1}^2 - {d_1}^2 = {r_2}^2 - {d_2}^2 \end{cases} \Rightarrow$$

$$\begin{cases} d_1 + d_2 = d \\ {d_2}^2 - {d_1}^2 = {r_2}^2 - {r_1}^2 \end{cases} \Rightarrow \begin{cases} d_1 + d_2 = d \\ (d_2 - d_1)(d_2 + d_1) = {r_2}^2 - {r_1}^2 \end{cases} \Rightarrow$$

$$(d_2 - d_1) = \frac{{r_2}^2 - {r_1}^2}{d} \qquad \Rightarrow \qquad d_2 = \frac{{r_2}^2 - {r_1}^2}{d} + d_1 = \frac{{r_2}^2 - {r_1}^2}{d} + d - d_2 \qquad \Rightarrow$$

$$2 \cdot d_2 = \frac{{r_2}^2 - {r_1}^2}{d} + d \qquad \Rightarrow \qquad d_2 = \frac{{r_2}^2 - {r_1}^2}{2 \cdot d} + \frac{d}{2} \tag{4.9}$$

Теперь найдем катет d_2 в треугольнике K_1M_2S по формуле (4.9).

Зная расстояния d и d_2, находим d_1 по формуле (4.1.3):

$$d_1 = d - d_2$$

Зная катет d_1 и гипотенузу r_1 найдем катет h в $\triangle M_1K_1S$ по формуле (4.10):

$$h = \sqrt{{r_1}^2 - {d_1}^2} \tag{4.10}$$

Далее находим координаты точки S через формулу деления отрезка в данном отношении (через координаты M_1 и M_2 и отношения k):

$$k = \frac{d_1}{d_2}$$

$$x_s = \frac{a_1 + k \cdot a_2}{1 + k} \tag{4.11}$$

$$y_s = \frac{b_1 + k \cdot b_2}{1 + k} \tag{4.12}$$

Далее найдем координаты обоих точек пересечения окружностей по формулам (4.13, 4.14, 4.15, 4.16):

$$x_1 = \frac{h}{d} \cdot (b_1 - b_2) + x_s \tag{4.13}$$

$$y_1 = \frac{h}{d} \cdot (a_2 - a_1) + y_s \tag{4.14}$$

$$x_2 = -\frac{h}{d} \cdot (b_1 - b_2) + x_s \tag{4.15}$$

$$y_2 = -\frac{h}{d} \cdot (a_1 - a_2) + y_s \tag{4.16}$$

Для выбора координат из полученных (x_1, y_1) и (x_2, y_2) проверим значения по координатам 3 маяка (на выбор).

Для вычисления координат точки на препятствии, видимую датчиком расстояния, сначала найдем угол α робота по формуле (4.17):

$$\alpha_{robot} = atan2(x_i - x_{i-1}; y_i - y_{i-1}) \tag{4.17}$$

после чего найдем угол дальномера β (с учетом того, что дальномер направлен влево по ходу движения робота) по формуле (4.18):

$$\beta = \alpha_{robot} + \frac{\pi}{2} \tag{4.18}$$

координаты точки на препятствии, вычисляемые с показаний дальномера, находим по формулам (4.19, 4.20):

$$x_o = x_{i\ robot} + d \cdot cos(\beta) \tag{4.19}$$

$$y_o = y_{i\ robot} + d \cdot sin(\beta) \tag{4.20}$$

где d - показания с дальномера до препятствия.

Результаты вычислений показаны на рис. 4.9

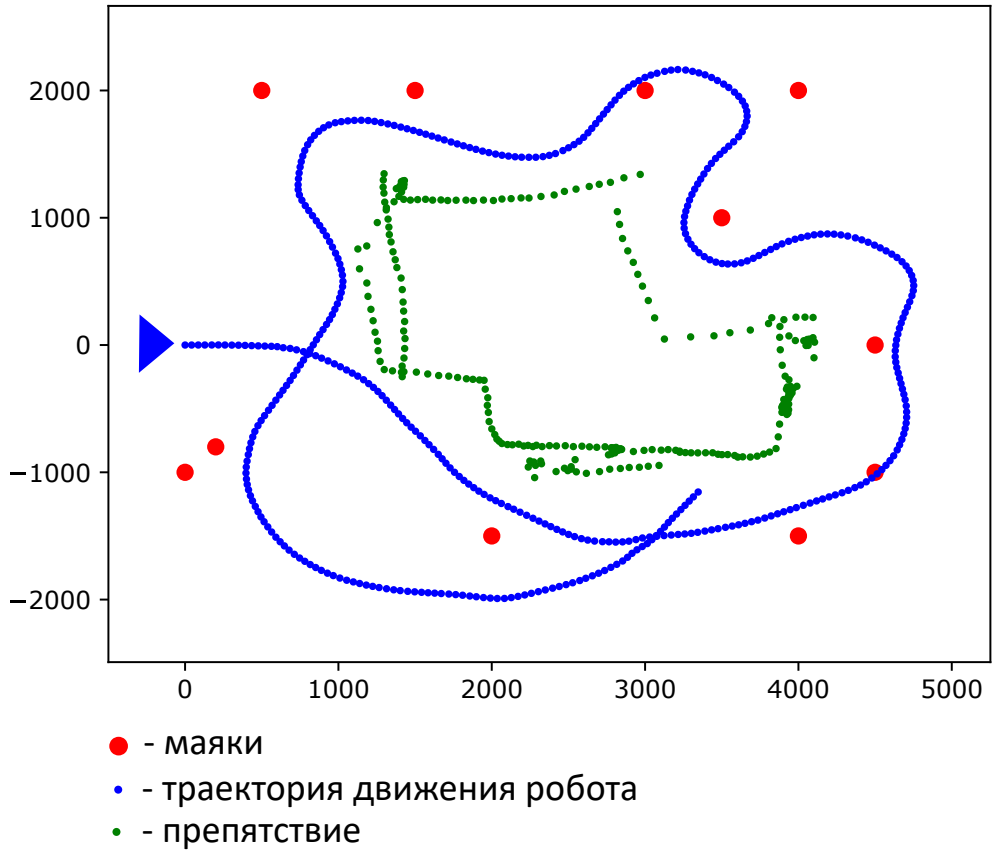


Рис. 4.9: Траектория движения робота и границы препятствия

По полученным координатам точек многоугольника находим площадь препятствия по формуле площади Гаусса (4.21):

$$\begin{aligned} A &= \frac{1}{2}\left|\sum_{i=1}^{n-1} x_i y_{i+1} + x_n y_1 - \sum_{i=1}^{n-1} x_{i+1} y_i - x_1 y_n\right| \\ &= \frac{1}{2}|x_1 y_2 + x_2 y_3 + \cdots + x_{n-1} y_n + x_n y_1 - x_2 y_1 - x_3 y_2 - \cdots - x_n y_{n-1} - x_1 y_n| \end{aligned} \tag{4.21}$$

где,

- A - площадь многоугольника
- n - количество сторон многоугольника

- (x_i, y_i) при $i = 1, 2, \ldots n$ – координаты вершин многоугольника

В ответ выводим полученную площадь препятствия.

Пример программы-решения

Ниже представлено решение на языке Python3

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


def gauss(polygon):
    # polygon в формате [(x1,y1),...(xi,yi)] или [[x1,y1], ... [xi,yi]]
    polygon.append(polygon[0])
    S = 0
    for i in range(len(polygon) - 1):
        S += polygon[i][0] * polygon[i + 1][1]       # Xi * Yi+1
        S -= polygon[i][1] * polygon[i + 1][0]       # Yi * Xi+1
    S = abs(S) * 0.5
    return int(S)


def circles(lst):
    a1, b1, r1, a2, b2, r2, a3, b3, r3 = lst
    # расстояние между центрами маяков M1 (a1, b1), M2 (a2, b2)
    d = math.sqrt((a2 - a1) ** 2 + (b2 - b1) ** 2)

    if d >= r1 + r2 or d < abs(r1-r2):
        return False

    d2 = (r2 ** 2 - r1 ** 2) / (2 * d) + (d / 2)
    d1 = d - d2

    h = math.sqrt(r1 ** 2 - d1 ** 2)
    k = d1 / d2

    xs = (a1 + k * a2) / (1 + k)
    ys = (b1 + k * b2) / (1 + k)

    x1 = (h / d) * (b1 - b2) + xs
    y1 = (h / d) * (a2 - a1) + ys

    x2 = -(h / d) * (b1 - b2) + xs
    y2 = -(h / d) * (a2 - a1) + ys

    test1 = int(math.sqrt((x1 - a3) ** 2 + (y1 - b3) ** 2))
    test2 = int(math.sqrt((x2 - a3) ** 2 + (y2 - b3) ** 2))

    if abs(test1 - r3) <= abs(test2 - r3):
        return (x1, y1)
    else:
        return (x2, y2)


data = sys.stdin.readlines()

K, N, dT = list(map(int, data[0].strip().split(' ')))
```

```
52  data.pop(0)
53  """
54  K  - кол-во маяков
55  N  - кол-во замеров
56  dT - пауза между замерами, мс
57  """
58
59  beacons, measures = [], []
60
61  for i in range(K):
62      beacons.append(list(map(float, data[i].strip().split(' '))))
63
64  for i in range(N):
65      measures.append(list(map(float, data[i + K].strip().split(' '))))
66
67  xy_obstacle, xy_robot = [], []
68  alpha = 0
69
70  for i in range(N):
71      b_idx = []
72      for b in range(len(beacons)):
73          if measures[i][b + 1] >= 0:
74              b_idx.append(b)
75          if len(b_idx) == 3:
76              break
77
78      b_info = []
79      for  idx in b_idx:
80          b_info.append(beacons[idx][1])
81          b_info.append(beacons[idx][2])
82          b_info.append(measures[i][idx + 1])
83
84      xy = circles(b_info)
85      xy_robot.append(xy)
86
87      if (measures[i][0] < 1000) and len(xy_robot) > 1: # замеры с дальномера
88          dY = xy_robot[-1][1] - xy_robot[-2][1]
89          dX = xy_robot[-1][0] - xy_robot[-2][0]
90          alpha = (math.atan2(dY, dX))
91
92          betta = alpha + math.radians(90)
93          xy = [xy_robot[-1][0] + measures[i][0] * math.cos(betta),
94              xy_robot[-1][1] + measures[i][0] * math.sin(betta)]
95
96          xy_obstacle.append(xy)
97
98  print(gauss(xy_obstacle))
```

Задача 4.1.4. Определение цветных цилиндров (15 баллов)

Робот, собранный по дифференциальный схеме и оснащенный дальномером, направленным прямо по ходу движения, перемещается по робототехническому полигону. На данном полигоне установлены цилиндры различного цвета. Робот также является цилиндром.

В процессе своего передвижения по полигону (см рис. 4.10) робототехническое устройство измерило расстояние до возникающих прямо препятствий и их цвет в шестнадцатиричном формате вида $RRGGBB$, где RR - 16тиричное число R состав-

ляющей данного элемента матрицы, GG и BB - 16тиричные числа G и B составляющих, соответственно.

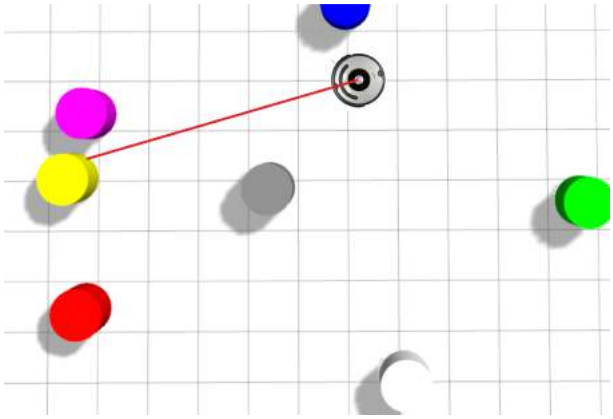


Рис. 4.10: Пример соревновательного полигона

Необходимо определить между какими цилиндрами робот не сможет проехать, если известен маршрут робота и показания датчика в процессе движения. Размер цилиндров и робота: 20 см в диаметре. Дальномер возвращает расстояние до цилиндра, считая от оси вращения робота, находящейся в центре между колёсами. Также гарантируется, что робот измерил каждый цилиндр не менее трех раз, и разница между этими измерениями составляет минимум 5 дуговых градусов (при измерении по дуге цилиндра).

Формат входных данных

Первая строка содержит 2 целых числа: N и dT:

- N — количество замеров ($10 \leqslant N \leqslant 10000$);
- dT — пауза между измерениями в мс ($0 \leqslant dT \leqslant 10000$).

Далее идёт N строк. Каждая строка имеет следующую структуру: Vel_{linear}, $Vel_{angular}$, $Distance$, $Color$, в которой все числа разделены пробелами, где:

- Vel_{linear} — линейная скорость в см/с, с которой движется робот в данный момент ($-100 \leqslant Vel_{linear} \leqslant 100$);
- $Vel_{angular}$ — угловая скорость в рад/мс, с которой движется робот в данный момент ($-10 \leqslant Vel_{angular} \leqslant 10$);
- $Distance$ — показания датчика расстояния в см в диапазоне “5–255” см, если предмет вне зоны видимости, то будет выведено 255;
- $Color$ — цвет обнаруженного циллиндра в шестнадцатиричном формате вида $RRGGBB$. В случае если цилиндр не обнаружен, будет выведено “000000”.

Значения скоростей и показания датчика являются вещественными числами. Моторы достигают скоростей мгновенно. Робот движется без проскальзывания. Значения цвета — целые числа.

Формат выходных данных

Одна строка, в которой указана пара цветов цилиндров в шестнадцатиричной записи через пробел, в порядке возрастания чисел. В случае если таких пар несколько, то каждую пару следует выводить в отдельной строке. Несколько строк следует

выводить в порядке возрастания первых чисел, а в случае их равенства, в порядке возрастания вторых.

Примеры

Примеры входных данных и ответов к ним можно найти по сслыке `http://bit.ly/2UoC22Z`.

Решение

Рассмотрим движение робота между моментами изменения скоростей. В этот момент линейная и угловая скорости постоянны и мы можем рассматривать движение робота, как показано на рис. 4.11.

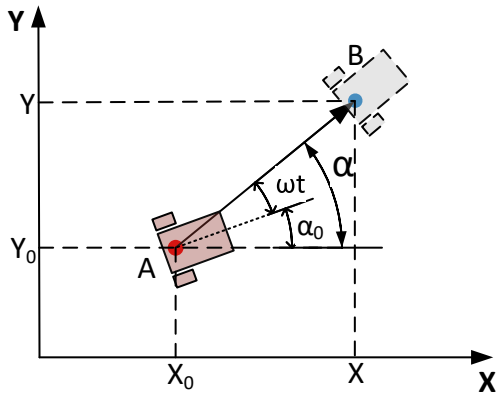


Рис. 4.11: Движение между измерениями скоростей

По условиям задачи нам известны угловая (ω) и линейная (v) скорости робота, а также время между замерами (t).
Угол поворота (α) вычисляем по формуле (4.22).
В начале движения робот находится в координатах ($x_0 = 0; y_0 = 0$) и направлен вдоль оси X, т.е. угол $\alpha_0 = 0$.

$$\alpha = \alpha_0 + w \cdot t \tag{4.22}$$

Длина пройденного пути (отрезок AB) находится по формуле (4.23):

$$l = v \cdot t \tag{4.23}$$

Координаты робота в точке B находятся по формулам (4.24) и (4.25):

$$x_i = x_0 + l \cdot cos(\alpha) \tag{4.24}$$

$$y_i = y_0 + l \cdot sin(\alpha) \tag{4.25}$$

После обработки всех данных получаем путь перемещения робота (рис. 4.12).

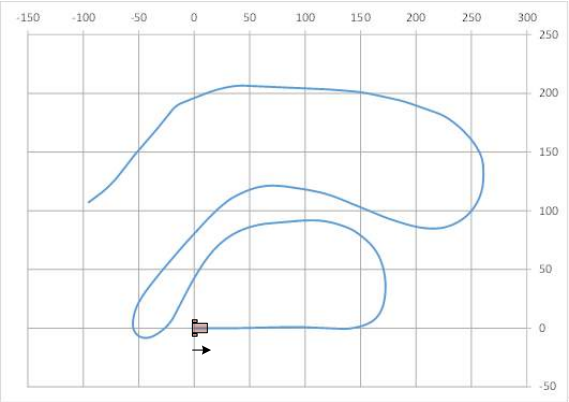


Рис. 4.12: Рассчитанный путь перемещения робота

Зная координаты робота и получаемое расстояние от дальномера (d) мы можем рассчитать координаты точек на окружностях цилиндров (с учетом того, что дальномер направлен прямо по ходу движения робота, см. рис. 4.13) по формулам (4.26) и (4.27):

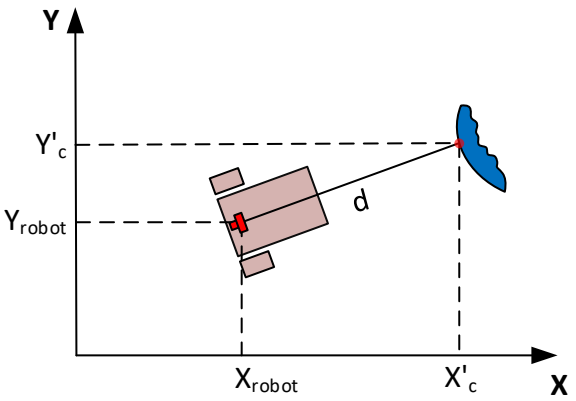


Рис. 4.13:

$$x'_c = x_{robot} + cos(\alpha) \cdot d \tag{4.26}$$

$$y'_c = y_{robot} + sin(\alpha) \cdot d \tag{4.27}$$

Координаты точек относятся к цилиндрам определенного цвета. Результат показан на рис. 4.14.

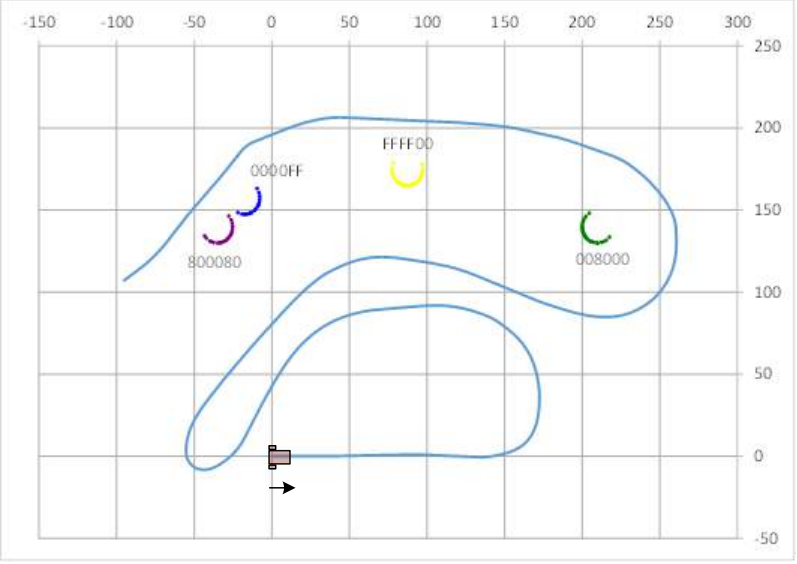


Рис. 4.14: Наборы точек, распределенные по цветам цилиндров

Сейчас нам известны координаты всех точек на окружности каждого цилиндра - найдем координаты центра каждого цилиндра. Для этого нам надо выбрать 3 точки из всего множества для каждого цилиндра. Например, две точки - самые крайние с обеих сторон, третья точка - посередине между ними (рис. 4.15). В примере на рис. 4.15 указан набор точек желтого цвета ($FFFF00$) (из рис. 4.14).

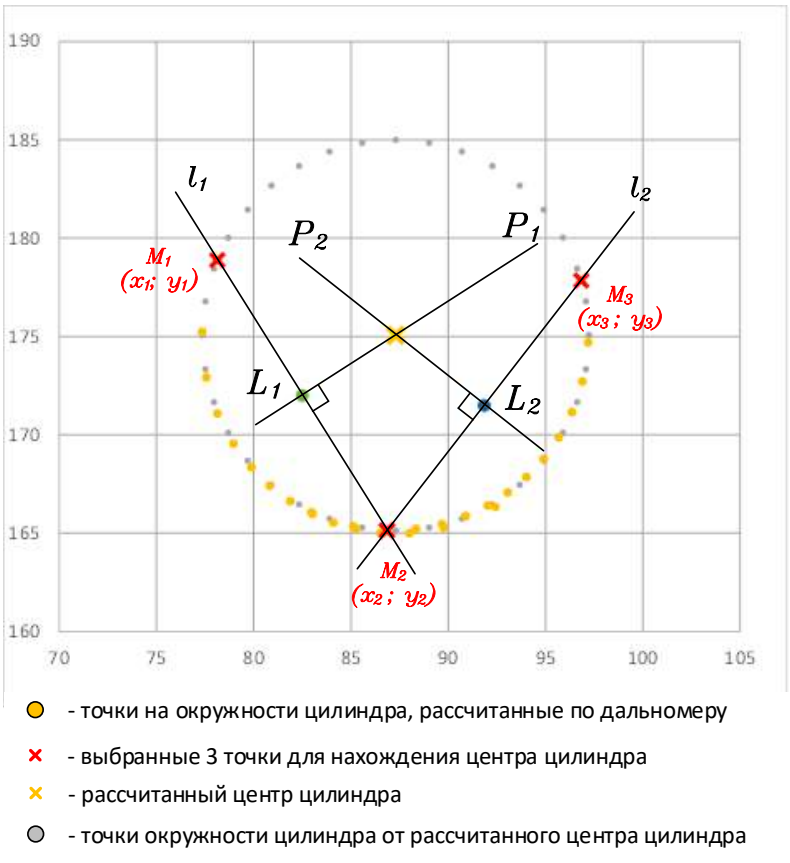


Рис. 4.15: Схема нахождения центра цилиндра по трем точкам

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 (4.28) и (4.28), (4.28) и (4.28):

$$\begin{aligned} k_{l_1} &= \frac{y_2 - y_1}{x_2 - x_1} \\ b_{l_1} &= y_2 - k_{l_1} \cdot x_2 \\ k_{l_2} &= \frac{y_3 - y_2}{x_3 - x_2} \\ b_{l_2} &= y_3 - k_{l_2} \cdot x_3 \end{aligned} \tag{4.28}$$

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 (4.29) и (4.29).

$$\begin{aligned} L_1 &= \left(\frac{x_1 + x_2}{2}; \frac{y_1 + y_2}{2} \right) \\ L_2 &= \left(\frac{x_2 + x_3}{2}; \frac{y_2 + y_3}{2} \right) \end{aligned} \tag{4.29}$$

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 (4.30) и (4.30), для P_2 - по формулам (4.30) и (4.30):

$$\begin{aligned} k_{P_1} &= -\frac{1}{k_{l_1}} \quad \Leftrightarrow \quad k_{P_1} = -\frac{x_2 - x_1}{y_2 - y_1} = \frac{x_1 - x_2}{y_2 - y_1} \\ b_{P_1} &= y_{L_1} - k_{P_1} \cdot x_{L_1} \\ k_{P_2} &= -\frac{1}{k_{l_2}} \quad \Leftrightarrow \quad k_{P_2} = -\frac{x_3 - x_2}{y_3 - y_2} = \frac{x_2 - x_3}{y_3 - y_2} \\ b_{P_2} &= y_{L_2} - k_{P_2} \cdot x_{L_2} \end{aligned} \tag{4.30}$$

Сейчас мы уже можем найти координаты точку центра цилиндра (или точку пересечения прямых P_1 и P_2) по формулам (4.31) и (4.31):

$$\begin{aligned} x &= \frac{b_{P_1} - b_{P_2}}{k_{P_2} - k_{P_1}} \\ y &= k_{P_1} \cdot x + b_{P_1} \end{aligned} \tag{4.31}$$

Результат расчета центров цилиндров показан на рис. (4.16):

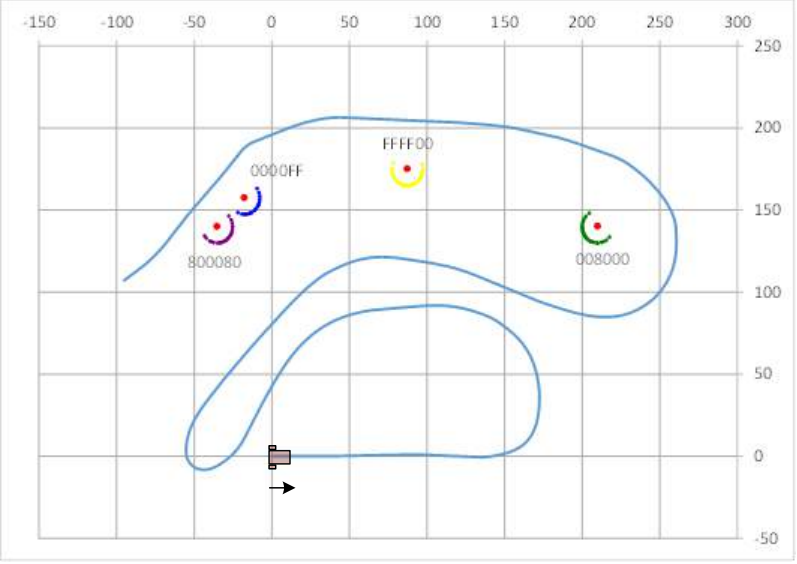


Рис. 4.16: Центры цилиндров

Далее проверяем попарно все цилиндры (каждый с каждым) на возможность проезда между ними робота. Для этого мы должны узнать расстояние между краями цилиндров по формуле (4.32). Для примера возьмем цилиндры с цветами 800080 и $0000FF$.

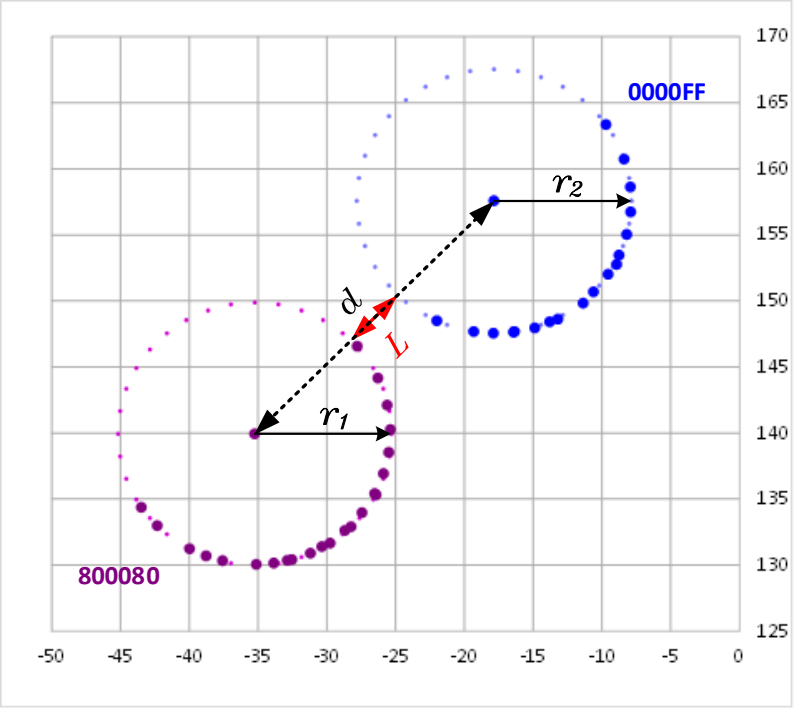


Рис. 4.17:

$$L = d - r_1 - r_2 \tag{4.32}$$

где d - расстояние между центрами цилиндров; r_1, r_2 - радиусы цилиндров. В нашем примере $d = 24.77$ см; $r_1 = 9.94$ см; $r_2 = 9.98$ см.

Если полученное расстояние L больше или равно диаметра робота ($L \geqslant D_{robot}$), значит робот сможет проехать между цилиндрами, иначе – нет.

В примере на рис. (4.17) расстояние между краями цилиндров $L = 4.85$ см, т.е. робот между этими цилиндрами не проедет, т.к. диаметр робота 20 см.

В ответ выводим пару цветов цилиндров, между которыми робот сможет проехать. Таких пар может быть несколько.

Ответ: 0000FF 800080

Пример программы-решения

Ниже представлено решение на языке Python3

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


def line_len(x1, y1, x2, y2):
    return math.sqrt((x2 - x1) ** 2 + (y2 - y1) ** 2)


def extreme_points(lst):
    max_len = 0
    xy = tuple()
    for i in range(len(lst)):
        x1, y1 = lst[i]
        for j in range(len(lst)):
            if line_len(x1, y1, *lst[j]) > max_len:
                max_len = line_len(x1, y1, *lst[j])
                xy = (x1, y1, *lst[j])
    return xy


data = sys.stdin.readlines()

N, dT = map(int, data[0].strip().split(' '))
"""
N  - кол-во замеров
dT - пауза между замерами, мс
"""
diameter = 20                                       # robots and cylinders, in cm
cylinder_colors = []
cylinder_dist_by_colors = {}

robot_xy = [[0, 0]]
alpha, robot_distance = 0, 0

for i in range(N):
    measures = data[i+1].strip().split(' ')       # v, w, dist_to_cylinder, color
    measures[:3] = list(map(float, measures[:3])) # v, w, dist_to_cylinder -> float()
    measures[3] = measures[3].rjust(6, '0')
    v, w, dist_to_cyl, color = measures

    alpha += w * dT
```

```
    robot_distance = v * dT

    r_x = robot_xy[-1][0] + robot_distance * math.cos(alpha)
    r_y = robot_xy[-1][1] + robot_distance * math.sin(alpha)

    robot_xy.append((r_x, r_y))

    if dist_to_cyl < 255 and color != '000000':
        cyl_x = robot_xy[-1][0] + dist_to_cyl * math.cos(alpha)
        cyl_y = robot_xy[-1][1] + dist_to_cyl * math.sin(alpha)

        if color not in cylinder_dist_by_colors:
            cylinder_dist_by_colors[color] = [(cyl_x, cyl_y)]
        else:
            cylinder_dist_by_colors[color].append((cyl_x, cyl_y))

unique_dist = {}
for key in cylinder_dist_by_colors:
    xy = []
    unique_dist[key] = []
    for x, y in cylinder_dist_by_colors[key]:
        str_xy = str(round(x, 4)) + ',' + str(round(y, 4))
        if str_xy not in xy:
            xy.append(str_xy)
            unique_dist[key].append([x, y])

cylinder_centers = []
for color in unique_dist:
    x1, y1, x2, y2 = extreme_points(cylinder_dist_by_colors[color])
    x3, y3 = cylinder_dist_by_colors[color][8]

    L1 = [(x1 + x2) / 2, (y1 + y2) / 2]
    L2 = [(x2 + x3) / 2, (y2 + y3) / 2]

    # P1
    k1 = (x1 - x2) / (y2 - y1)
    b1 = L1[1] - k1 * L1[0]

    # P2
    k2 = (x2 - x3) / (y3 - y2)
    b2 = L2[1] - k2 * L2[0]

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

    cylinder_centers.append([x, y, color])

results = []
for i in range(len(cylinder_centers)):
    for j in range(1, len(cylinder_centers)):
                if i != j and i < j:
                    x1, y1 = cylinder_centers[i][0:2]
                    clr1 = cylinder_centers[i][2]
                    x2, y2 = cylinder_centers[j][0:2]
                    clr2 = cylinder_centers[j][2]

                    dist = math.sqrt((x1 - x2) ** 2 + (y1 - y2) ** 2)
                    if dist - diameter < diameter:
                        results.append(sorted([clr1, clr2]))

```

```
103 for i in results:
104     print(*i)
```

Задача 4.1.5. Определения расстояния до препятствия (5 баллов)

Камера статично установлена на высоте h мм и отклонена на α° от горизонта. Она имеет разрешение $height \times width$ и угол обзора β°. Поясняющая картинка представлена на рисунке 4.18.

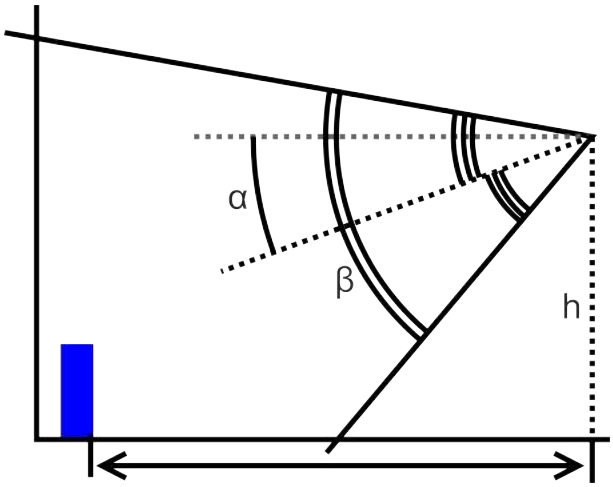


Рис. 4.18: Схема установки камеры

Необходимо найти расстояние до предмета, установленного на полигоне. Гарантируется, что предмет находится в поле зрения камеры, является однотонным. Его минимальная площадь на изображении равна $s\%$. Также известно, что помимо требуемого предмета в камеру попадает поверхность полигона и(или) борта. Поверхность полигона не однородная, а содержащая различные цвета в хаотическом порядке.

Пример изображения представлен на рис. 4.19,

Рис. 4.19: Пример изображения с камеры

Формат входных данных

Первая строка входного файла содержит 6 чисел: h, α, β, $height$, $width$, s:

- h — высота установки камеры в мм ($10 \leqslant h \leqslant 100$);

- α — угол в градусах, под которым расположена камера относительно горизонта ($0° \leqslant \alpha \leqslant 45°$);
- β — угол обзора камеры в градусах ($30° \leqslant \alpha \leqslant 180°$);
- $height$ — разрешение изображения по высоте ($10 \leqslant height \leqslant 2000$);
- $width$ — разрешение изображения по ширине ($10 \leqslant width \leqslant 2000$);
- s — минимальная площадь предмета в % от всего изображения, которое измеряется в пикселях ($1 \leqslant s \leqslant 100$).

На следующих $height$ строках расположено изображение размером $height \times width$ в виде шестнадцатиричных чисел. Данные числа имеют следующий вид: $RRGGBB$, где RR - 16тиричное число R составляющей данного элемента матрицы, GG и BB - 16тиричные числа G и B составляющих, соответственно.

Все числа являются целыми.

Формат выходных данных

Одна строка, в которой указано число — расстояние до предмета в мм. Допускается погрешность в 10 мм.

Примеры

Примеры входных данных и ответов к ним можно найти по сслыке `http://bit.ly/2FCbgAU`.

Решение

Составим схему по данным из условий задачи (рис. 4.20):

1. Высота установки камеры, h
2. Угол отклонения камеры от линии горизонта, α
3. Угол обзора камеры, β
4. Разрешение камеры по высоте, $height$

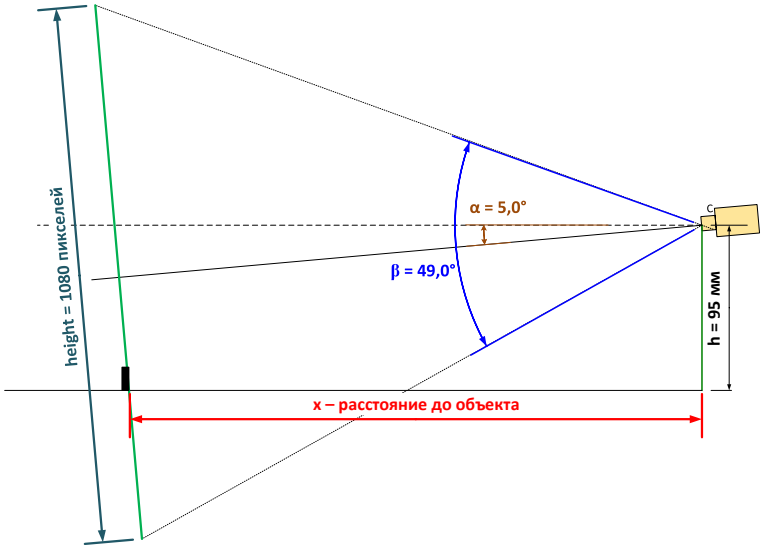


Рис. 4.20:

Теперь вполне очевидно, что искомое значение x - это катет b прямоугольного треугольника $\triangle ABC$ (рис. 4.21)

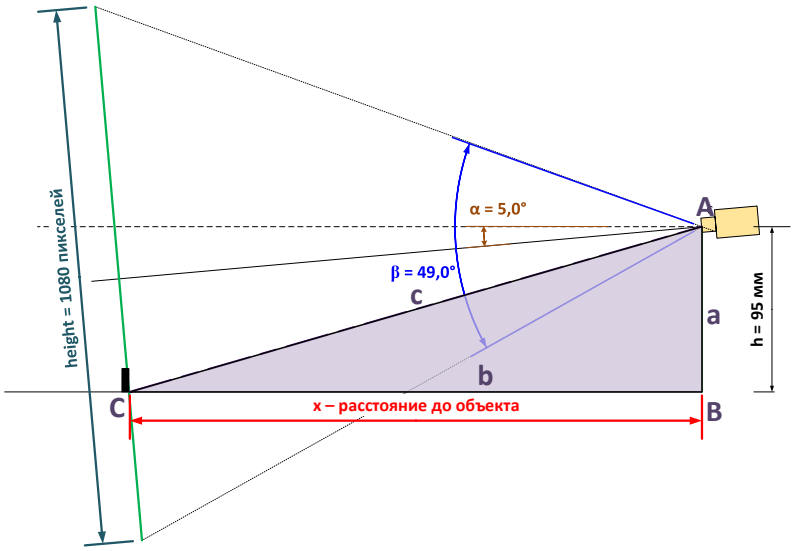


Рис. 4.21:

Как вариант, найти длину катета b можно через $tg(\angle CAB)$, тогда пусть $\angle CAB = \omega$.
Для нахождения угла ω нам надо найти углы θ и γ.
Для нахождения угла θ нам надо знать расстояние x_1. В нашем случае, расстояние x_1 - это высота в пикселях от нижней границы изображения до нижней границы предмета на изображении. см. рис.4.22.

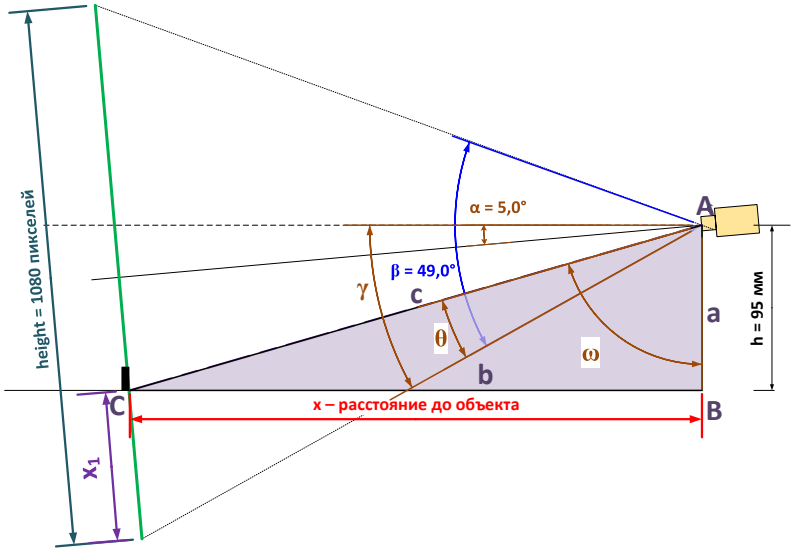


Рис. 4.22:

Нахождение расстояния x_1: По условиям задачи известно, что искомый объект на изображении - "однотонный". Алгоритм поиска однотонного изображения заключается в поиске близлежащих пикселей одинакового цвета. Для этого используется построчное сканирование изображения с применением алгоритма BFS к каждому непросмотренному пикселю. При этом надо вести учет уже просмотренных ранее пикселей через BFS. Начальный пиксель находится в левом верхнем углу.

После обработки всего изображения сравниваем площади полученных объектов и находим объект с площадью, максимально приближенной к задаваемой в условии задачи.

Далее у этого объекта берем значения пикселя с ближайшими координатами к нижней границе изображения, рис. 4.23.

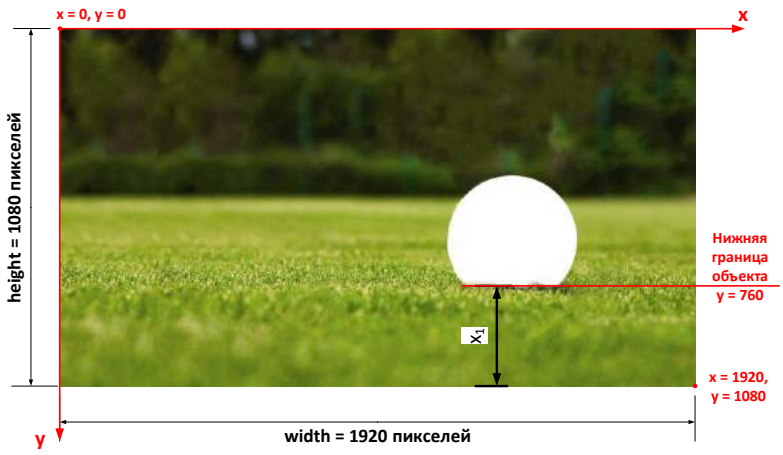


Рис. 4.23:

Находим расстояние x_1:

$x_1 = height$ – максимальная координата Y объекта $= 1080 - 720 = 320$ пикселей

Угол θ найдем через пропорцию:

$$\frac{\beta}{\theta} = \frac{height}{x_1} \qquad \Rightarrow \qquad \theta = \frac{x_1 \cdot \beta}{height} \qquad \Rightarrow \qquad \theta = \frac{320 \cdot 49}{1080} = 14.52^\circ$$

Угол γ в нашем случае - это сумма угла α и половины угла β:

$$\gamma = \alpha + \frac{\beta}{2} \qquad \Rightarrow \qquad \gamma = 5 + \frac{49}{2} = 29.5^\circ$$

И, наконец, найдем угол ω:

$$\omega = 90 - \gamma + \theta \qquad \Rightarrow \qquad \omega = 90 - 29.5 + 14.52 = 75.02^\circ$$

Теперь найдем X, но предварительно переведем градусы в радианы:

$$\omega' = \frac{\omega \cdot \pi}{180} \qquad \Rightarrow \qquad \omega' = \frac{95 \cdot \pi}{180} = 1,309346005 \text{ радиан}$$

$$x = h \cdot tg(\omega') \qquad \Rightarrow \qquad x = 95 \cdot tg(1,309346005) = 355 \text{ мм}$$

Ответ: расстояние от камеры до объекта равно 355 мм

Пример программы-решения

Ниже представлено решение на языке Python3

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


def find_obj2(pic):
    row_max = len(pic)
    col_max = len(pic[0])
    visited = [[False for j in range(col_max)] for i in range(row_max)]

    colors = {}
    for row in range(row_max - 1, 0, -1):
        for col in range(col_max):
            if not visited[row][col]:
                clr = pic[row][col]
                key = clr + ',' + str(row) + ',' + str(col)
                colors[key] = [(row, col)]
                queue = [(row, col)]
                while len(queue) > 0:
                    r, c = queue.pop(0)  # row, col
                    if not visited[r][c]:
                        visited[r][c] = True
                        colors[key].append((r, c))

```

```
25                          if r + 1 < row_max:
26                              if pic[r + 1][c] == clr and not visited[r + 1][c]:
27                                  queue.append((r + 1, c))
28                          if r - 1 >= 0:
29                              if pic[r - 1][c] == clr and not visited[r - 1][c]:
30                                  queue.append((r - 1, c))
31                          if c + 1 < col_max:
32                              if pic[r][c + 1] == clr and not visited[r][c + 1]:
33                                  queue.append((r, c + 1))
34                          if c - 1 >= 0:
35                              if pic[r][c - 1] == clr and not visited[r][c - 1]:
36                                  queue.append((r, c - 1))
37
38                  if len(colors[key]) < 10:
39                      colors.pop(key)
40                  else:
41                      # добавляем площадь цвета от общего размера картинки
42                      colors[key].insert(0, len(colors[key]) / (pic_h * pic_w)
43                          * 100)
44
45      # выбираем цвет, у которого минимальная дельта к площади из условий (sq)
46      # for clr in colors:
47      #     print (clr, len(colors[clr]))
48
49      min_delta = float('inf')
50      for key in colors:
51          delta = abs(colors[key][0] - sq)
52          if  delta < min_delta:
53              key_clr = key
54              min_delta = delta
55
56      max_x = 0
57      for xy in colors[key_clr][1:]:
58          if xy[0] > max_x:
59              max_x = xy[0]
60
61      return max_x
62
63
64  def avg(arr):
65      return sum(arr) / len(arr)
66
67  virtual = True
68
69  results = []
70  data = sys.stdin.readlines()
71
72  cam_h, cam_alpha, cam_beta, pic_h, pic_w, sq = list(map(int,
73      data[0].strip().split(' ')))
74  data.pop(0)
75
76  """
77  cam_h                  - высота установки камеры, мм
78  camAlpha          - угол камеры относительно горизонта, градусы
79  camBeta                  - угол камеры, градусы
80  picH                  - высота изображения, пиксели
81  picW                  - ширина изображения, пиксели
82  sq                   - минимальная площадь предмета, % от общей площади картинки
83  """
84
```

```
picRaw, picDec = [], []

for row in range(pic_h):
    picRaw.append([])
    lineRaw = list(data[row].strip().split(' '))
    picRaw[row] = lineRaw

object_bottom = find_obj2(picRaw) # pixels from picture_bottom to object_bottom
angle = 90 - cam_alpha - cam_beta/2 + (pic_h - object_bottom)/pic_h * cam_beta

result = cam_h * math.tan(math.radians(angle))
print (round(result))
```

Задача 4.1.6. Определение вектора перемещения робота в заданном виде (20 баллов)

Робот, собранный по дифференциальной схеме, оборудован камерой. Камера установлена так, что смотрит вниз перпендикулярно поверхности, по которой движется робот. Известны характеристики камеры такие, как разрешение ($height \times width$), угол обзора α, высота расположения над плоскостью движения. Верхний край кадра находится дальше от робота, чем нижний край кадра.

Есть последовательность кадров, сделанных камерой через равные промежутки времени. Всего кадров было сделано N.

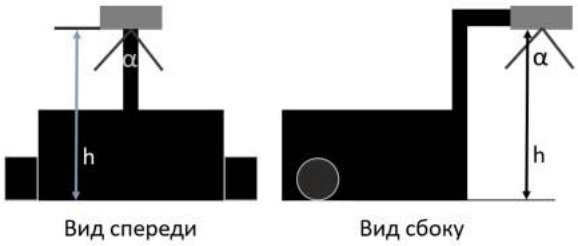


Рис. 4.24: Вид робота. Поясняющая картинка

Зная, что робот движется с постоянной линейной (v) и постоянной угловой (ω) скоростью, необходимо определить перемещение по X и по Y в мм.

Формат входных данных

Входной файл состоит из нескольких строк.

Первая строка содержит 5 целых чисел: N, h, α, $height$, $width$:

- N — количество замеров камерой ($2 \leqslant N \leqslant 16$);
- h — высота установки камеры в мм ($10 \leqslant h \leqslant 2000$);
- α — угол обзора камеры в градусах ($15° \leqslant \alpha \leqslant 175°$);
- $height$ — разрешение изображения по высоте ($5 \leqslant height \leqslant 16$);
- $width$ — разрешение изображения по ширине ($5 \leqslant width \leqslant 16$).

Далее расположено N строк, на каждой из которых расположено изображение размером $height \times width$ в виде шестнадцатиричных чисел слева направо, сверху

вниз. Данные числа имеют следующий вид: $RRGGBB$, где RR - 16тиричное число R составляющей данного элемента матрицы, GG и BB — 16тиричные числа G и B составляющих соответственно.

Все числа являются целыми.

Формат выходных данных

Вывести пару вещественных чисел с точностью до целых — перемещение по X и по Y в мм в данном порядке через пробел.

Примеры

Примеры входных данных и ответов к ним можно найти по сслыке `http://bit.ly/2EP1c5g`.

Решение

Для примера рассмотрим схему установки камеры (рис. 4.25), снимки с камеры (рис. 4.26) и набор данных:

- Количество замеров камерой, $N = 3$
- Высота установки камеры, $h = 100$ мм
- Угол обзора камеры, $\alpha = 30°$
- Разрешение снимка камеры по высоте, $height = 15$ пикселей
- Разрешение снимка камеры по ширине, $width = 15$ пикселей

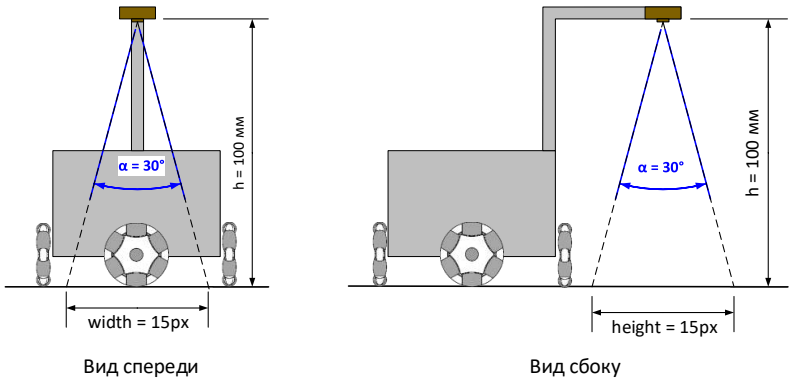


Рис. 4.25: Схема установки камеры с исходными данными

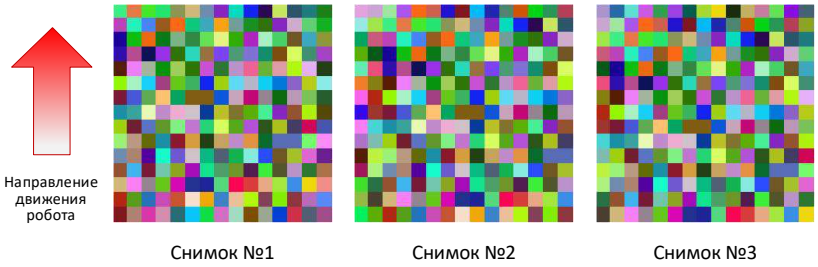


Рис. 4.26: Пример снимков (1 цв.квадрат = 1 пиксель) с камеры размером 15x15 пикселей

Для решения задачи сначала вычисляем перемещения робота через смещения пикселей на снимках (рис. 4.27). Сравниваем по два смежных снимка: $_{i-1i}$. После обработки всех снимков получаем конечное смещение в пикселях.

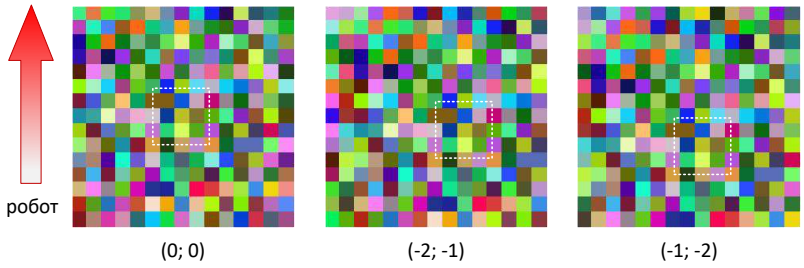


Рис. 4.27: Вычисление перемещения робота

Для пересчета смещения из пикселей в мм рассмотрим схему на рис. 4.28 и переведем полученные относительные координаты $(x; y)$ в мм.

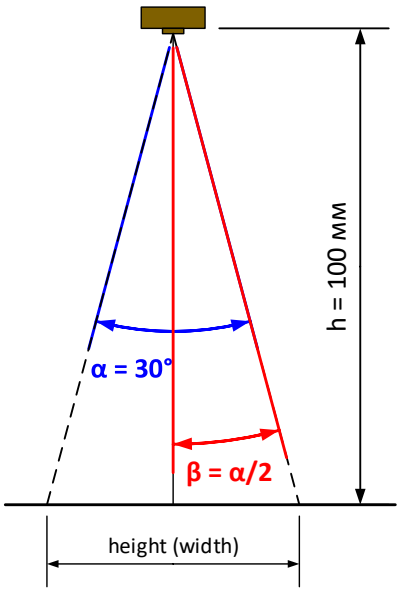


Рис. 4.28:

Вычислим размерность 1 пикселя для ширины и высоты снимка (формулы 4.33 и 4.34).

$$pixel\_width\_mm = \frac{h \cdot tg(\beta) \cdot 2}{width} \Rightarrow \frac{100 \cdot tg(0.2618) \cdot 2}{15} = 3.57 \text{ мм} \tag{4.33}$$

$$pixel\_height\_mm = \frac{h \cdot tg(\beta) \cdot 2}{height} \Rightarrow \frac{100 \cdot tg(0.2618) \cdot 2}{15} = 3.57 \text{ мм} \tag{4.34}$$

Переводим пиксели в мм:

$$x = x \cdot 3.57 = -1 \cdot 3.57 = -3.6(\text{мм})$$
$$y = y \cdot 3.57 = -2 \cdot 3.57 = -7.1(\text{мм})$$

Ответ: -3.6 -7.1

Пример программы-решения

Ниже представлено решение на языке Python3

```
1  # -*- coding: utf-8 -*-
2
3  import math
4  import sys
5
```

```
6  def check(mat1, mat2):  #функция для нахождения смещения между двумя кадрами
7      maximum = 0
8      max_y = 0
9      max_x = 0
10     for i in range(-pic_h + 1, pic_h):
11         for j in range(-pic_w + 1, pic_w):
12             max_this = 0
13             for y in range(pic_h):
14                 for x in range(pic_w):
15                     if (y + i) >= 0 and (x + j) >= 0 and (y + i) < pic_h and
16                        (x + j) < pic_w:
17                         if mat1[y][x] == mat2[y + i][x + j]:
18                             max_this += 1
19             if max_this > maximum:
20                 max_x = -j
21                 max_y = -i
22                 maximum = max_this
23     return [maximum, max_x, max_y]
24
25 data = sys.stdin.readlines()
26
27 shots_n, cam_h, cam_alpha, pic_h, pic_w = list(map(int, data[0].strip().split(' ')))
28 data.pop(0)
29 '''
30 #shots_n    - кол-во замеров
31 #h          - высота установки камеры, мм
32 #cam_alpha  - угол обзора камеры, градусы
33 #pic_h      - высота изображения
34 #pic_w      - ширина изображения
35 '''
36 shots = []
37
38 for i in range(shots_n):
39     shots.append([]) # очередной кадр
40     shot_raw = data[i].strip().split(' ')
41     # кадр в виде одномерного массива пикселей в hex
42     shot_dec = list(map(lambda hx: int(hx,16), shot_raw))
43     # кадр в виде одномерного массива пикселей в dec
44     for r in range(0, pic_h * pic_w, pic_w):
45         shots[-1].append(shot_dec[r:r + pic_w])
46         # очередная строка пикселей в кадре i
47
48 x, y = 0, 0
49 for i in range(1, shots_n):
50     temp = check(shots[i-1], shots[i])
51
52     x += int(temp[1])
53     y += int(temp[2])
54
55 x = math.tan(math.radians(cam_alpha / 2)) * 2 * cam_h / pic_w * x
56 y = math.tan(math.radians(cam_alpha / 2)) * 2 * cam_h / pic_h * y
57 print(str(round(x, 1)) + " " + str(round(y, 1)))
```

Задача 4.1.7. Распознавание ARTag маркера (10 баллов)

Для определения своего местоположения квадрокоптер использует камеру, снимающую поверхность, над которой перемещается робот. Изображение с камеры приходит в управляющую программу в виде набора $height \times width$ точек, где каждая

точка закодирована в RGB-формате.

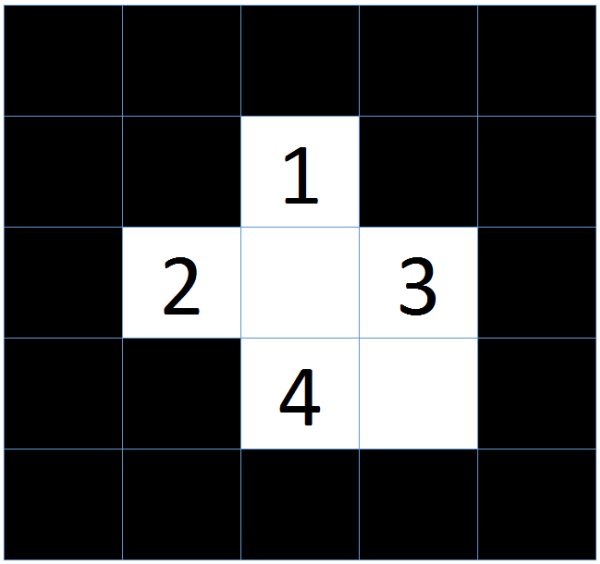


Рис. 4.29: Нумерация битов в маркера

На поверхность нанесены ARTag маркеры (`https://inside.mines.edu/~whoff/courses/EENG512/lectures/other/ARTag.pdf`). Элементы маркера, 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 в он обозначает 1, если белый, то 0 при этом самый первый элемент — старший бит закодированного числа. Элементы пронумерованы сверху вниз, слева направо (см. рис. 4.29).

Например, на маркере с рис. 4.30 закодировано число 0011_2, что эквивалентно 3_{10}.

Поскольку квадрокоптер не перемещается постоянно параллельно поверхности, то изображения ARTag маркера, получаемые с камеры, получаются в виде неправильного выпуклого четырехугольника, а непостоянные условия освещенности изменяют фокус, тон и добавляют блики на изображение. Также квадрокоптеру не всегда удается полностью захватить маркер, и необходимо обрабатывать данные с нескольких снимков.

Поскольку направление запуска квадрокоптера заранее неизвестно, то ориентация маркеров заранее неизвестна, но его изображение таково, что оно по каждой из осей X, Y, Z относительно оптической оси камеры не превышает 25 градусов.

Напишите программу для определения закодированного в маркере числа.

Формат входных данных

Входной файл состоит из нескольких строк.

Первая строка содержит 3 целых числа: N, $height$, $width$:

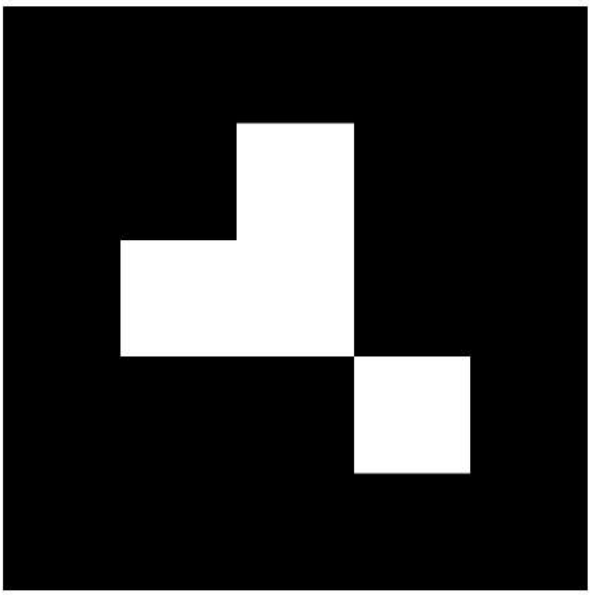


Рис. 4.30: Маркер с закодированным значением - 0011_2

- N — количество замеров камерой ($1 \leqslant N \leqslant 100$);
- $height$ — разрешение изображения по высоте ($10 \leqslant height \leqslant 1000$);
- $width$ — разрешение изображения по ширине ($10 \leqslant width \leqslant 1000$).

Далее расположено N строк на каждой расположено изображение размером $height \times width$ ввиде шестнадцатиричных 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

Вывести одно целое число в десятичной системе счисления, которое было закодированное на маркере. В случае если невозможно определить число, следует вывести -1.

Примеры

Примеры входных данных и ответов к ним можно найти по сслыке `http://bit.ly/2DHww5w`.

Решение

Декомпозиция задачи

Для каждого кадра:

1. Перевод цветного изображения в ч/б (состоит из "0"и "1")
2. Распознавание метки ARTag
    (a) Нахождение углов многоугольника (до 8)

(b) если углов больше 4 - "достраиваем"срезанные углы

(c) Нахождение центра метки, он же - бит четности (parity bit)

(d) Нахождение ключевого бита (key bit)

(e) "Сборка"двоичного числа

(f) Сверка с битом четности

(g) Перевод в десятичную систему счисления

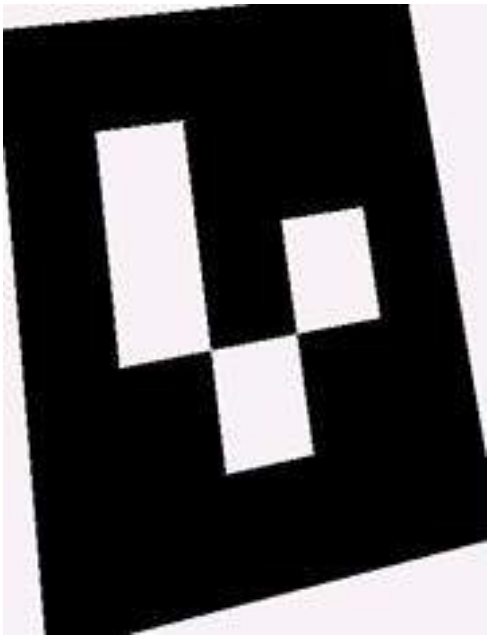


Рис. 4.31: Исходное изображение метки ARTag

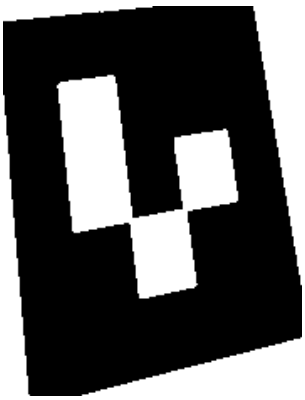


Рис. 4.32: Изображение в ч/б варианте

С алгоритмами перевода изображения в черно-белый вариант можно ознакомиться в презентации "Алгоритмические основы технического зрения"(`http://robolymp.ru/forum/index.php?PAGE_NAME=list&FID=116`).

Находим координаты углов n-угольника. В нашем случае получается 8-угольник:

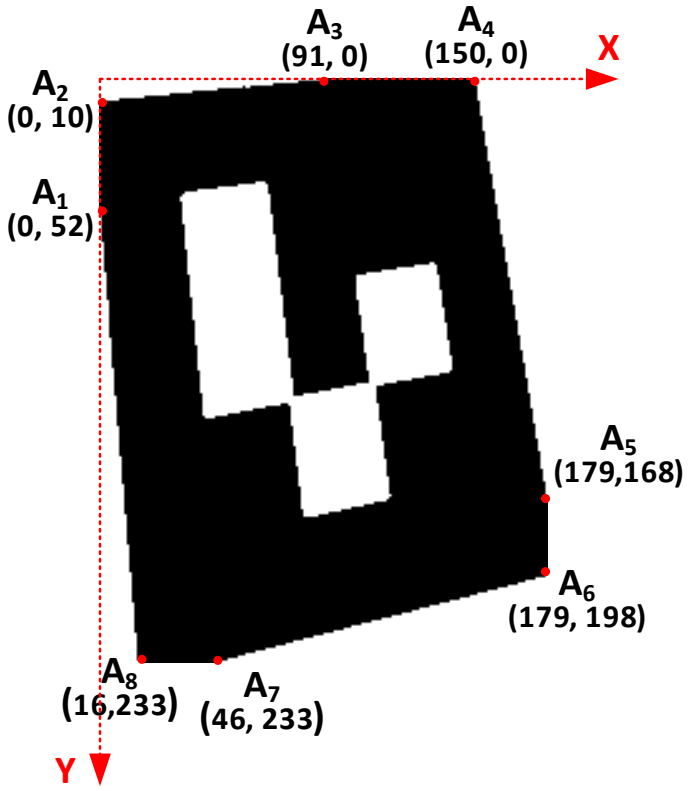


Рис. 4.33: Координаты углов метки ARTag

	A_1	A_2	A_3	A_4	A_5	A_6	A_7	A_8
X	0	0	91	150	179	179	46	16
Y	52	10	0	0	168	198	233	233

Таблица 4.2: Координаты углов метки ARTag

Получается, что все 4 угла 'срезаны' значит нам надо их 'достроить'.

Каждый угол образуется пересечением отрезков близлежащих сторон метки. Например, ∠A (верхний левый) образуется пересечением отрезков $(A_1; A_8)$ и $(A_2; A_3)$, т.е. для нахождения координат угла нам достаточно найти точку пересечения прямых.

Исходя из линейной функции $y = k \cdot x + b$ найдем коэффициенты $k_{1,2}$ и $b_{1,2}$ для каждого отрезка:

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

Для отрезка $(A_1; A_8)$:

$$k_1 = \frac{233 - 52}{16 - 0} = 11.31; \qquad b_1 = 233 - 11.31 \cdot 16 = 52.04$$

Для отрезка $(A_2; A_3)$:

$$k_2 = \frac{0 - 10}{91 - 0} = -0.11; \qquad b_2 = 0 + 0.11 \cdot 91 = 10.01$$

Теперь находим координаты $x; y$ ∠A:

$$x = \frac{b_1 - b_2}{k_2 - k_1} = \frac{52.04 - 10.01}{-0.11 - 11.31} = -3.68$$
$$y = k_1 \cdot x + b_1 = 11.31 \cdot -3.68 + 52.04 = 10.42$$

Аналогичным образом находим остальные углы и получаем координаты 4х угольника:

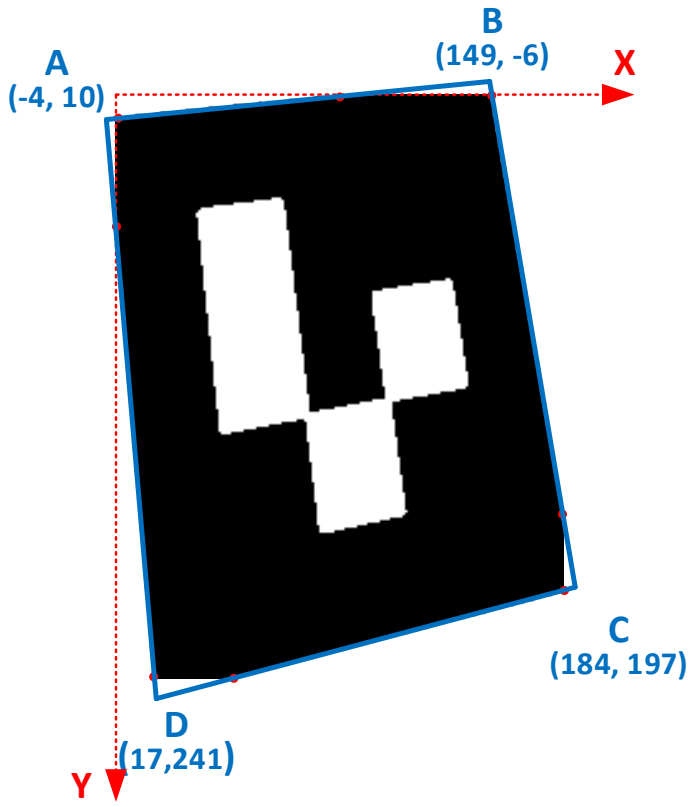


Рис. 4.34: Координаты полученного 4-х угольника

Далее находим центр метки, он же будет "битом четности". В нашем случае центр метки - это точка пересечения диагоналей метки: $AC \cap BD$, в результате получаем координаты центра: $O(90; 104)$.

После чего находим центры сторон 4х-угольника: AB, BC, CD, DA. Для этого мы разделим отрезок каждой стороны на 2 равные части. Разберем на примере стороны AB:

$$x_{AB} = \frac{x_1 + x_2}{2} = \frac{-4 + 149}{2} = 72.5$$
$$y_{AB} = \frac{y_1 + y_2}{2} = \frac{10 - 6}{2} = 2$$

Аналогичным способом находим координаты середин остальных сторон, в результате получаем (рис.4.35).

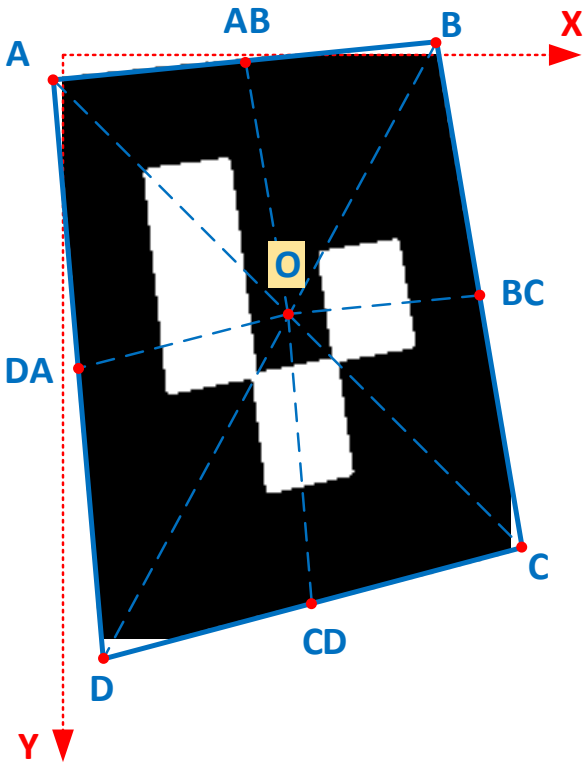


Рис. 4.35: Разделение сторон периметра метки пополам

	O	AB	BC	CD	DA
X	90	73	167	101	7
Y	104	-2	96	219	126

Таблица 4.3: Координаты центра метки и середин сторон периметра метки

Теперь каждый отрезок от точки O до точки на периметре 4х-угольника делим на 2.5 части. Почему на 2.5? В нашем случае маркер представляет квадрат из 5х5 клеток и центр находится в середине клетки, поэтому расстояние от центра метки до любой точки на краю метки составляет 2.5 клетки (рис. 4.36).

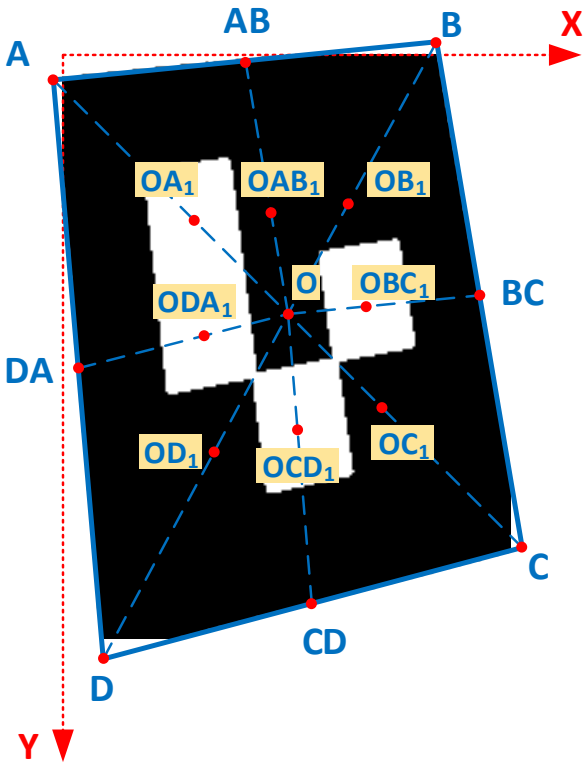


Рис. 4.36: Полученные точки в центрах клеток

По полученным точкам OA_1, OB_1, OC_1, OD_1 находим "ключевой бит по которому определятся начало отсчета двоичного числа на метке. В нашем случае "ключевой бит"находится в точке OA_1.

По точкам OAB_1, OBC_1, OCD_1, ODA_1 составляем двоичный код. Старший разряд двоичного числа является самым верхним от расположения "ключевого бита".

Расположение информационных битов, в зависимости от "ключевого бита показаны на рис. 4.1.7 и 4.1.7.

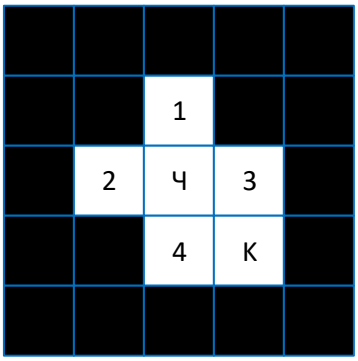


Рис. 4.37: Общая схема метки ARTag

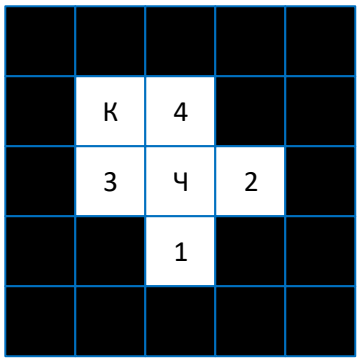


Рис. 4.38: Расположение битов для нашего примера

В нашем случае получается двоичное число: 0001
Сверяем двоичное число с "битом четности": если в двоичном числе количество "1"нечетное, значит "бит четности"должен быть равен "1 если же количество "1"в двоичном числе четное, то "бит четности"должен быть равен "0".

Если проверка на четность прошла успешно, то принимаем ответ как правильный, если проверка не прошла, значит обрабатываем следующий кадр.

В нашем случае проверка прошла успешно, значит переводим полученное число из двоичной системы в десятичную и принимаем как правильный ответ, получается: 1.

Если кадров несколько, то лучше все их распознать и сравнить полученные результаты.

Ответ: на метке закодировано число 1.

Пример программы-решения

Ниже представлено решение на языке Python3

```
# -*- coding: utf-8 -*-
import sys
import math
import numpy as np

def cross_find(coords):
    # coords  - массив с координатами 4х угольника (ABCD)
    x1, y1 = coords[0]
    x2, y2 = coords[2]
    x3, y3 = coords[1]
    x4, y4 = coords[3]

    x = -(((x1 * y2 - x2 * y1) * (x4 - x3) - (x3 * y4 - x4 * y3) * (x2 - x1)) / (
                (y1 - y2) * (x4 - x3) - (y3 - y4) * (x2 - x1)))
    y = ((y3 - y4) * (-x) - (x3 * y4 - x4 * y3)) / (x4 - x3)

    return [int(x), int(y)]


def lineToSegments(begin, end, segments = 2):
    # begin, end - массивы с координатами отрезка (начало, конец).
```

```
22      # начало, как правило - центр
23      x0 = round((end[0] - begin[0]) / segments)
24      y0 = round((end[1] - begin[1]) / segments)
25
26      points = [begin]
27      for i in range(1, segments):
28          points.append([])
29          points[i].append(points[i - 1][0] + x0)
30          points[i].append(points[i - 1][1] + y0)
31      points.append([end])
32      return points
33
34
35  def rgb2clr(rgb):
36      #return (rgb[0] * 256 ** 2 + rgb[1] * 256 + rgb[2])
37      if type(rgb) == int:                # один цвет, значит переводим в серый
38          rgb = [rgb]
39          for i in range(2):
40              rgb.append(rgb[0])
41      return int(sum([rgb[i] * 256**(2-i) for i in range(len(rgb))]))
42
43
44  def clr2rgb(clr):
45      r = (clr & 0xFF0000) >> 16
46      g = (clr & 0x00FF00) >> 8
47      b = (clr & 0x0000FF)
48      return (r, g ,b)
49
50
51  def blur(pic_in): # div, offset):
52      blur5 = [
53          [0.000789, 0.006581, 0.013347, 0.006581, 0.000789],
54          [0.006581, 0.054901, 0.111345, 0.054901, 0.006581],
55          [0.013347, 0.111345, 0.225821, 0.111345, 0.013347],
56          [0.006581, 0.054901, 0.111345, 0.054901, 0.006581],
57          [0.000789, 0.006581, 0.013347, 0.006581, 0.000789]
58      ]
59      blur3 = [
60          [0.054901, 0.111345, 0.054901],
61          [0.111345, 0.225821, 0.111345],
62          [0.054901, 0.111345, 0.054901]
63      ]
64
65      pic_out, pic_blur = [], []
66      size = 3                    # "window" size
67      side = size //2
68
69      pic_h, pic_w = len(pic_in), len(pic_in[0])
70      for i in range(side, pic_h - side):
71          pic_out.append([])
72          for j in range(side, pic_w - side):
73
74              # pixels - 2D список цветов пикселей "окна" в каналах R,G,B
75              # очередное "окно" из пикселей 3*3 для применения фильтра
76              pixels = []
77              for p_y in range(i-side, i+side+1):
78                  pixels.append([])
79                  for p_x in range(j-side, j+side+1):
80                      pixels[-1].append(clr2rgb(pic_in[p_y][p_x]))
81
```

```
82              # pixels_RGB - 2D список списков цветов по-канально [R], [G], [B]
83              pixels_RGB = []
84              for ii in range(3):
85                  pixels_RGB.append([[rgb[ii] for rgb in row] for row in pixels])
86
87              pic_blur = []
88              for channel in range(3):
89                  pic_blur.append([])
90                  for ii in range(size):
91                      pic_blur[-1].append([])
92                      for jj in range(size):
93                          if size == 3:
94                              pic_blur[-1][-1].append(pixels_RGB[channel][ii][jj]
95                                  * blur3[ii][jj])
96                          elif size == 5:
97                              pic_blur[-1][-1].append(pixels_RGB[channel][ii][jj]
98                                  * blur5[ii][jj])
99
100             pixels_RGB_blur = [int(sum([sum(row) for row in ch_blur]))
101                 for ch_blur in pic_blur]
102             pic_out[-1].append(rgb2clr(pixels_RGB_blur))
103
104     return pic_out
105
106
107 def pic_to_RGB (pic_in):
108     pic_out = []
109     for row in range(len(pic_in)):
110         pic_out.append([])
111         for col in range(len(pic_in[0])):
112             pic_out[-1].append(clr2rgb(pic_in[row][col]))
113     return pic_out
114
115
116 def rgb2grey(pic_in, mode):
117     RGB, pic_out = [], []
118     pic_h, pic_w = len(pic_in), len(pic_in[0])
119     for i in range(pic_h):
120         pic_out.append([])
121         for j in range(pic_w):
122             RGB = clr2rgb(pic_in[i][j])
123             if mode == 1:                # "V from HSV", V = max(r, g, b)
124                 result = max(RGB)
125             elif mode == 2:              # через Y ' from Y'UV(средневзвешенное) вариант 1
126                 # result = Math.round((0.299 * RGB[0]) + (0.587 * RGB[1])
127                 # + (0.114 * RGB[2]))
128                 result = round((0.2126 * RGB[0]) + (0.7152 * RGB[1]) + (0.0722 * RGB[2]))
129             elif mode == 3:  # через Y ' from Y' UV(средневзвешенное) вариант 2
130                 clr = pic_in[i][j]
131                 # tmp = ((((clr >> 16) & 0xff) * 76) + (((clr >> 8) & 0xff) * 150)
132                 # + ((clr & 0xff) * 29)) >> 8
133                 tmp = ((((clr >> 16) & 0xff) * 76) + (((clr >> 8) & 0xff) * 150) + ((clr & 0xff) * 29
134                 result = tmp << 16 | tmp << 8 | tmp  # r | g | b
135             elif mode == 4:          # Поиск ближайшей точки нейтральной оси:
136                                      # L = (max(R, G, B) + min(R, G, B)) / 2
137                 result = round((max(RGB) + min(RGB)) / 2)
138             elif mode == 5:          # Среднее арифметическое компонент R, G, B
139                 result = round(sum(RGB) / 3)
140
141             #print (result,rgb2clr(result))
```

```
142             pic_out[-1].append(rgb2clr(result))
143
144     return pic_out
145
146
147 def rgb2bw(pic_in, method = 'niblack'):
148     #pic_in_RGB = pic_to_RGB(pic_in)
149
150     pic_h, pic_w = len(pic_in), len(pic_in[0])
151     pic_out = [[16777215 for _ in range(pic_w)] for _ in range(pic_h)]
152
153     delta = 0
154     dot_side = 3
155     radius = int(dot_side/2)      # pixels from center 'big dot' to each side
156
157     if method == 'niblack' or method == 'sauvola':
158         '''
159         Niblack:
160         T = m + k * s
161         m - mean of local area pixels
162         k - 0.2     - value by author
163         s - standart deviation of local pixel area
164
165         Sauvola:
166         T = m * (1 - k * 1 - S / R)
167         m - mean of pixels under window area
168         S - dynamic range of variance
169         k - may be in the range of 0-1 (?)
170         k=0.5 and R=128         - values by author
171
172         '''
173         for row in range(delta + radius, pic_h - delta - radius):
174             for col in range(delta + radius, pic_w - delta - radius):
175                 pixels = []
176                 for p_r in range(dot_side):
177                     for p_c in range(dot_side):
178                         pixels.append(pic_in[row - radius + p_r][col - radius + p_c])
179                 #print (row,col,list(map(hex, pixels)))
180                 mean = sum(pixels)/len(pixels)
181                 if method == 'niblack':
182                     sd = standart_deviation(pixels)
183                     T = int(mean + sd * 0.2)
184                     #print (mean, T)
185                 else:
186                     S = standart_variance(pixels)
187                     T = int(mean * (1 - 0.5 * (1 - S / 128)))
188
189                 # pix = 16777215 if pic_in[row][col] >= T else 0
190                 if pic_in[row][col] < T:
191                     pic_out[row][col] = 0
192
193     elif method == 'threshold':
194         for row in range(pic_h):
195             for col in range(pic_w):
196                 if pic_in[row][col] < 1500000:
197                     pic_out[row][col] = 1
198                 else:
199                     pic_out[row][col] = 0
200
201     return pic_out
```

```
202
203
204 def standart_deviation(nums):
205     mean = sum(nums)/len(nums)
206     return ((1/len(nums) * sum([(num - mean) **2 for num in nums])/(len(nums))) ** 0.5)
207
208
209 def standart_variance(nums):
210     mean = sum(nums)/len(nums)
211     return (sum([(num - mean) **2 for num in nums])/len(nums))
212
213
214 def k_b(P1, P2):
215     x1, y1 = P1
216     x2, y2 = P2
217     k = (y2 - y1) / (x2 - x1)
218     b = y2 - k * x2
219     return (k, b)
220
221 def mean_xy(corner1, corner2):
222     # вычисление средних координат x, y двух точек
223     mean_x = int(corner1[0] + corner2[0]) / 2
224     mean_y = int(corner1[1] + corner2[1]) / 2
225     return (mean_x, mean_y)
226
227
228 def bias_xy(corner, m_xy):
229     # отклонение координат x, y  от среднего значения
230     return (abs(corner[0] - m_xy[0]), abs(corner[1] - m_xy[1]))
231
232
233 def close_corners(corners):
234     BIAS = 10                                  # допустимая погрешность
235     abcd = [()] * 4
236
237     meanXY = mean_xy(corners[0],corners[1])
238     bias = bias_xy(corners[0], meanXY)
239     print (corners[6], corners[0])
240     if sum(bias) > BIAS and (corners[6] != corners[0] and corners[2] != corners[1]):
241         k1, b1 = k_b(corners[6], corners[0])
242         k2, b2 = k_b(corners[2], corners[1])
243         x = (b1 - b2) / (k2 - k1)
244         y = k1 * x + b1
245     else:
246         x, y = meanXY
247     abcd[0] = (int(x), int(y))
248
249     meanXY = mean_xy(corners[2], corners[3])
250     bias = bias_xy(corners[3], meanXY)
251     if sum(bias) > BIAS and (corners[1] != corners[2] and corners[5] != corners[3]):
252         k1, b1 = k_b(corners[1], corners[2])
253         k2, b2 = k_b(corners[5], corners[3])
254         x = (b1 - b2) / (k2 - k1)
255         y = k1 * x + b1
256     else:
257         x, y = meanXY
258     abcd[1] = (int(x), int(y))
259
260     meanXY = mean_xy(corners[4], corners[5])
261     bias = bias_xy(corners[4], meanXY)
```

```
    if sum(bias) > BIAS and (corners[3] != corners[5] and corners[7] != corners[4]):
        k1, b1 = k_b(corners[3], corners[5])
        k2, b2 = k_b(corners[7], corners[4])
        x = (b1 - b2) / (k2 - k1)
        y = k1 * x + b1
    else:
        x, y = meanXY
    abcd[2] = (int(x), int(y))

    meanXY = mean_xy(corners[7], corners[6])
    bias = bias_xy(corners[7], meanXY)
    if sum(bias) > BIAS and (corners[4] != corners[7] and corners[0] != corners[6]):
        k1, b1 = k_b(corners[4], corners[7])
        k2, b2 = k_b(corners[0], corners[6])
        x = (b1 - b2) / (k2 - k1)
        y = k1 * x + b1
    else:
        x, y = meanXY
    abcd[3] = (int(x), int(y))

    return abcd


def get_corners(pic_in):
    corners = [(-1, -1)] * 8
    # left_up x 2, right_up x2, right_down x2, left_down x2
    #pic_h, pic_w = len(pic_in), len(pic_in[0])

    out_l = out_r = False
    for row in range(pic_h):                        # Up -> Down
        for col_l in range(pic_w):                  # L -> R
            if pic_in[row][col_l] == 1 and not out_l:
                corners[0] = (row, col_l)
                out_l = True

        for col_r in range(pic_w-1, -1, -1):        # L <- R
            if pic_in[row][col_r] == 1 and not out_r:
                corners[1] = (row, col_r)
                out_r = True

    out_l = out_r = False
    for row in range(pic_h-1, -1, -1):              # Up <- Down
        for col_l in range(pic_w):                  # L -> R
            if pic_in[row][col_l] == 1 and not out_l:
                corners[4] = (row, col_l)
                out_l = True

        for col_r in range(pic_w-1, -1, -1):        # L <- R
            if pic_in[row][col_r] == 1 and not out_r:
                corners[5] = (row, col_r)
                out_r = True

    out_u = out_d = False
    for col in range(pic_w):                        # Left -> Right
        for row_u in range(pic_h):                  # U -> D
            if pic_in[row_u][col] == 1 and not out_u:
                corners[6] = (row_u, col)
                out_u = True

        for row_d in range(pic_h-1, -1, -1):        # D <- U
```

```
322             if pic_in[row_d][col] == 1 and not out_d:
323                 corners[7] = (row_d, col)
324                 out_d = True
325 
326     out_u = out_d = False
327     for col in range(pic_w-1, -1, -1):              # Left <- Right
328         for row_u in range(pic_h):                  # U -> D
329             if pic_in[row_u][col] == 1 and not out_u:
330                 corners[2] = (row_u, col)
331                 out_u = True
332 
333         for row_d in range(pic_h-1, -1, -1):        # D <- U
334             if pic_in[row_d][col] == 1 and not out_d:
335                 corners[3] = (row_d, col)
336                 out_d = True
337 
338     print(corners)
339     if sum(map(len, corners)) == 0:                  # не найдено ни одного угла
340         return -1
341     elif no_corner(corners[0], corners[1]) or no_corner(corners[2], corners[3]) or \
342         no_corner(corners[4], corners[5]) or no_corner(corners[6], corners[7]):
343         return -1                                    # нету одного из углов
344     else:
345         return close_corners(corners)
346 
347 
348 def find_corners2(pic_in, delta = 0):
349 
350     looked = []
351 
352     def check_xy(yx):
353         if pic_h - delta > yx[0] >= 0 + delta and pic_w - delta > yx[1] >= 0 + delta:
354             return yx not in looked and pic_in[yx[0]][yx[1]] == 1
355         else:
356             return False
357 
358 
359     def find_x(y0x0, direction = 1):
360         print (y0x0)
361         y0, x0 = y0x0
362         prev_yx = (0, 0)
363         dX, dY = prev_yx
364 
365         while True:
366             y, x = y0, x0
367             if check_xy((y - direction, x)):
368                 y -= direction
369                 dY += 1
370             elif check_xy((y, x + direction)):
371                 x += direction
372                 dX += 1
373             elif check_xy((y + direction,x)):
374                 y += direction
375                 dY += 1
376 
377             if (y, x) not in perimeter:
378                 perimeter.append((y, x))
379 
380             if x != x0 and dY > 0:
381                 dX, dY = 0, 0
```

```
382
383             if x == x0 and dY > 1:
384                 looked.pop()
385                 return (prev_yx)
386
387             #print('yx0:',y0, x0, '  yx:', y, x, '  ', y0 == y, x0 == x, ' dYX:',dY, dX)
388             looked.append((y, x))
389             prev_yx = (y0, x0)
390             if y0 == y and x0 == x:
391                 x += direction
392             y0, x0 = y, x
393
394
395     def find_y(y0x0, direction = 1):
396         y0, x0 = y0x0
397         prev_yx = (0, 0)
398         dX, dY = prev_yx
399
400         while True:  # from up to down
401             y, x = y0, x0
402             if check_xy((y, x + direction)):
403                 x += direction
404                 dX += 1
405             elif check_xy((y + direction, x)):
406                 y += direction
407                 dY += 1
408             elif check_xy((y, x - direction)):
409                 x -= direction
410                 dX += 1
411
412             if (y, x) not in perimeter:
413                 perimeter.append((y, x))
414
415             if y0 != y and dX > 0:
416                 dX, dY = 0, 0
417
418             if y == y0 and dX > 1:
419                 looked.pop()
420                 return prev_yx
421
422             #print('yx0:',y0, x0, '   yx:', y, x, '   ', y0 == y, x0 == x,
423             # '  dYX:',dY, dX, '  black[y,x]:',pic_in[y][x],' look:',
424             # (y,x) in looked)
425
426             looked.append((y, x))
427             prev_yx = (y0, x0)
428             if y0 == y and x0 == x:
429                 if (y,x) not in looked:
430                     y += direction
431                 else:
432                     break
433             y0, x0 = y, x
434
435     pic_h, pic_w = len(pic_in), len(pic_in[0])
436     start = (-1, -1)
437
438     for y in range(delta, pic_h - delta):
439         for x in range(delta, y + 1): # (delta):
440             if pic_in[x][y - x + delta] == 1:
441                 start = (x, y - x + delta)
```

```
442                 break
443         if sum(start) >= 0:
444             break
445 
446     perimeter = [start]
447     corners0 = [start]
448     corners0.append(find_x(corners0[-1]))
449     corners0.append(find_y(corners0[-1]))
450     print (corners0)
451     corners0.append(find_x(corners0[-1], -1))
452     find_y(corners0[-1], -1)                        # для заполнения точек периметра
453 
454 #    for i in perimeter:
455 #        print (*i)
456 
457     corners = corners0.copy()
458 
459     def corner_at_border(xy):
460         return xy[0] == pic_h-1 or xy[0] == 0 or xy[1] == pic_w - 1 or xy[1] == 0
461 
462     if not corner_at_border(corners[0]) or not corner_at_border(corners[1]) or \
463         not corner_at_border(corners[2]) or not corner_at_border(corners[3]):
464         c = 1
465         for (y0, x0) in corners0:
466             dX = sum([1 for y, x in perimeter if y == y0]) - 1
467             dY = sum([1 for y, x in perimeter if x == x0]) - 1
468             yx = (y0, x0)
469 
470             if dY > dX:
471                 if (y0 + dY, x0) in perimeter:
472                     yx = (y0 + dY, x0)
473                 else:
474                     yx = (y0 - dY, x0)
475             elif dX > dY:
476                 if (y0, x0 + dX) in perimeter:
477                     yx = (y0, x0 + dX)
478                 else:
479                     yx = (y0, x0 - dX)
480             corners.insert(c, yx)
481             c += 2
482 
483         #for c in corners:
484         #    print (*c)
485 
486     return corners
487 
488 
489 def no_corner(c1, c2):
490     if sum(c1) < 0 or sum(c2) < 0:
491         return True
492 
493 
494 def decode_artag(abcd):
495     center = cross_find(abcd)               # center of ARTag
496     parity = pic_bw[center[0]][center[1]]
497 
498     '''
499        K (keys) cells
500        -------------
501        | 3 |   | 0 |
```

```
502         -------------
503         |   |   |   |
504         -------------
505         | 2 |   | 1 |
506         -------------
507     '''
508     # отрезки от центр к углам
509     # из полученных точек берем только [1], т.к. [0] - это центр метки
510     K = []
511     for i in range(4):
512         K.append(tuple(map(int, lineToSegments(center, abcd[i], 3)[1])))
513     #print('K',K)
514 
515     KEY = -1
516     for i in range(4):
517         y, x = K[i]
518         if pic_bw[y][x] == 0:
519             KEY = i
520             break
521     print ('key',KEY)
522     if KEY == -1:                                   # не найдена "ключевая" клетка
523         return -1
524 
525     '''
526         bin's cells
527         -------------
528         |   | 3 |   |
529         -------------
530         | 2 |   | 0 |
531         -------------
532         |   | 1 |   |
533         -------------
534     '''
535 
536     bins = []
537     for i in range(4):
538         x, y = map(int, lineToSegments(K[i], K[(i+1) % 4])[1])
539         bins.append(pic_bw[x][y])
540     print ('bins', bins)
541     states = [(2, 3, 1, 0), (3, 2, 0, 1), (0, 3, 1, 2), (1, 0, 2, 3)]
542 
543     bin_num = ''
544     for i in range(4):
545         bin_num += str(bins[states[KEY][i]])
546 
547     # четное "1" - бит Ч в "0",     нечетое "1" - бит Ч в "1"
548     if bin_num.count('1') % 2 == 0 and parity == 0 or \
549             bin_num.count('1') % 2 != 0 and parity == 1: # проверка на бит четности
550         return int(bin_num, 2)
551     else:
552         return -1      # не прошел контроль четности
553 
554 
555 def bfs_edges(pic_in):
556     def check_range(y, x):
557         if pic_h > y >= 0 and pic_w > x >= 0:
558             return True
559         else:
560             return False
561 
```

```
562     def check_white(y, x):
563         for i in range(3):
564             for j in range(3):
565                 if i == j == 1:
566                     break
567                 if check_range(y - 1 + i, x - 1 + j):
568                     if pic_in[y - 1 + i][x - 1 + j] == 0:           # white pixel
569                         return True
570         return False
571 
572     def check_black(y, x):
573         bl = 0
574         for i in range(3):
575             for j in range(3):
576                 if check_range(y - 1 + i, x - 1 + j):
577                     if pic_in[y - 1 + i][x - 1 + j] == 1:           # black pixel
578                         bl += 1
579         if bl < 7:
580             return True
581         else:
582             return False
583 
584     pic_h, pic_w = len(pic_in), len(pic_in[0])
585     start = (-1, -1)
586     for i in range(pic_h):
587         for j in range(i):
588             if pic_in[j][i-j] == 1:
589                 start = (j, i-j)
590                 break
591         if sum(start) >= 0:
592             break
593 
594     visited = [[False for _ in range(pic_w)] for _ in range(pic_h)]
595     queue = [start]
596     edges = []
597     while len(queue) > 0:
598         y, x = queue.pop(0)
599         if not visited[y][x]:
600             visited[y][x] = True
601             if check_white(y, x):
602                 edges.append((y, x))
603             elif (x == 0 or y == 0) and check_black(y, x):
604                 edges.append((y, x))
605             elif (x == pic_w-1 or y == pic_h-1) and check_black(y, x):
606                 edges.append((y, x))
607 
608             for i in range(3):
609                 for j in range(3):
610                     if not i == j == 1:
611                         yy = y - 1 + i
612                         xx = x - 1 + j
613                         if check_range(yy, xx) and not visited[yy][xx]
614                             and pic_in[yy][xx] == 1:      # black pixel
615                             queue.append((yy, xx))
616 
617         if (y, x) == start and len(edges) > 1:
618             break
619 
620     # оставляем только периметр из точек
621     perimeter = []
```

```
622     looked = []
623     queue = [edges[0]]
624     while len(queue) > 0:
625         y0, x0 = queue.pop(0)
626         if not (y0, x0) in looked:
627             looked.append((y0, x0))
628             perimeter.append((y0, x0))
629             for y, x in edges:
630                 dY = abs(max(y0, y) - min(y0, y))
631                 dX = abs(max(x0, x) - min(x0, x))
632                 if (dY <= 1 and dX <= 1) and (y, x) not in looked and (y, x)
633                     not in queue:
634                     queue.append((y, x))
635 
636     return perimeter
637 
638 
639 path = "C:\\_docs\\!RobonesT\\!!Проекты\\Разбор НТИ ИРС\\zad7\\"
640 fileIn = path + "datasets\\0"
641 fileOut = path + "zad7_out.txt"
642 
643 with open(fileIn) as f:
644     data = f.readlines()
645 
646 # data = sys.stdin.readlines()
647 shots_n, pic_h, pic_w = list(map(int, data[0].strip().split(' ')))
648 data.pop(0)
649 """
650 #shots_h    - кол-во замеров камерой
651 #pic_h             - ширина изображения, пиксели
652 #pic_w             - высота изображения, пиксели
653 """
654 
655 pics_dec = []
656 for picN in range(shots_n):
657     pics_dec.append([])
658     for row in range(pic_h):
659         line = list(map(lambda h: int(h, 16),
660             data[row + picN * pic_h].strip().split(' ')))
661         pics_dec[picN].append(line)
662 
663 print ('get file done')
664 
665 pic_grey = rgb2grey(pics_dec[0], 2)
666 hist = {}
667 for row in pic_grey:
668     for clr in row:
669         clr_rgb0 = clr2rgb(clr)[0]
670         if str(clr_rgb0) not in hist:
671             clr_txt = str(clr_rgb0).rjust(3,'0')
672             hist[clr_txt] = 1
673         else:
674             hist[clr_txt] += 1
675 
676 for k, v in hist.items():
677     print(k, v)
678 
679 pic_blur = blur(pic_grey)
680 
681 pic_bw = rgb2bw(pic_blur, 'threshold')
```

```
682
683 with open(fileOut, mode='w') as f:
684     for line in pic_bw:
685         f.write(','.join(list(map(str,line))) + '\n')
686
687 print(find_corners2(pic_bw, 5))
688
689 #z = np.array(pic_bw, dtype = int)
690
691 #for i in z:
692 #print (sum(z[:,41]))
693 #ed = bfs_edges(pic_bw)
694 #print (ed)
695 #corners = [(-1,-1)] * 8
696 #print (sorted(ed))
697 #corners[0] = ed[0]
698 #for y in ed:
699 #    print (*y)
700
701
702 """
703
704 results = []
705 for i in range(shots_n):
706     pic_bw = rgb2bw(pics_dec[i],'threshold')
707     cs = get_corners(pic_bw)
708     if cs == -1:
709         res = -1
710     else:
711         res = decode_artag(cs)
712     results.append(res)
713     print ('result', results)
714
715 # vertical output
716 '''
717 spc = '16711680,' * pic_w                                    # красный цвет
718 spc = spc[:-1] + '\n'
719 with open(fileOut, mode='w') as f:
720 for i in pics_dec[0]:
721     #for j in range(len(pics_dec[0])):
722        #f.write(','.join(list(map(str, pics_dec[i][j]))) + '\n')
723    f.write(str(i).replace("[","").replace("]","") + '\n')
724 f.write(spc)
725 f.write(spc)
726 '''
727
728 # horizontal output
729 '''
730 spc = '16711680, 16711680, 16711680'
731 pics_out = []
732 for i in range(pic_h):
733     line = ''
734     for shot in range(shots_n):
735         line += ','.join(list(map(str, pics_dec[shot][i])))
736         line += ' ,' + spc
737     pics_out.append(line)
738 '''
739
740 """
741
```

```
#with open(fileOut, mode='w') as f:
 #   for line in pic_bw:
 #       f.write(','.join(list(map(str,line))) + '\n')
```

Задача 4.1.8. Определение лабиринта (10 баллов)

Роботы в количестве N, собранные по дифференциальной схеме, оснащёны тремя дальномерами, направленными налево, прямо и направо относительно направления движения робота. Роботы движутся одновременно из заранее известных секторов по неизвестному лабиринту, размером $K \times M$ секторов.

Между секторами могут быть стены, нахождение препятствия, внутри сектора недопустимо. Отсчёт координат секторов начинается с $(0; 0)$ в левом верхнем углу, X возрастает вправо, Y –вниз.

В процессе своих перемещений они получают значения с дальномеров, позволяющих узнать структуру лабиринта. В результате полученных данных необходимо понять в какие сектора полигона может попасть первый робот, без учёта нахождения на поле других роботов. В случае если данное значение невозможно определить, следует вывести количество секторов, посещённых в процессе движения данным роботом.

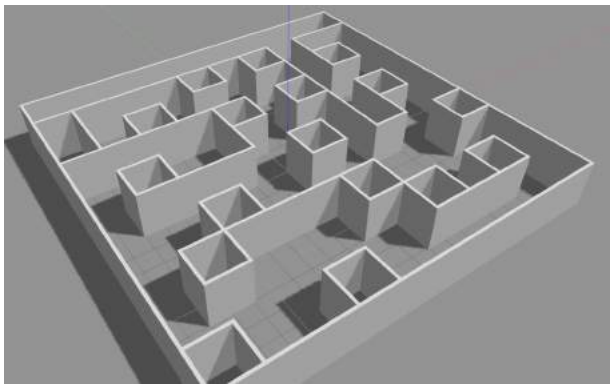


Рис. 4.39: Пример соревновательного полигона

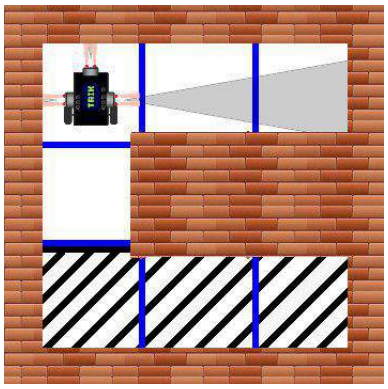


Рис. 4.40: Расположение датчиков

Формат входных данных

Первая строка содержит 4 целых числа: N, K, M и i:

- N — количество роботов на поле ($2 \leqslant N \leqslant 5$);
- K — ширина (по оси X) лабиринта в секторах ($4 \leqslant K \leqslant 20$);
- M — длина (по оси Y) лабиринта в секторах ($4 \leqslant M \leqslant 20$);
- i — количество показаний каждого робота.

Далее идет N строк, содержащие координаты старта каждого робота, а также направление робота при старте, т.е. одна строка имеет следующую структуру: x_s, y_s, dir, где:

- x_s — координаты старта данного робота по оси X;
- y_s — координаты старта данного робота по оси Y;
- dir — направление робота при старте:
  - U — робот направлен вверх;
  - L — робот направлен влево;
  - D — робот направлен вниз;
  - R — робот направлен вправо;

Далее идет N блоков по i строк. На каждой строке находится действие робота и показания всех датчиков после этого действия через пробел: $Movement$, D_l, D_f, D_r:

- $Movement$ — выполненное действие робота:
  - F — проезд робота в следующий по ходу движения сектор;
  - L — поворот робота в данном секторе налево;
  - R — поворот робота в данном секторе направо;
- D_l — показания датчика расстояния, направленного влево:
  - 1 — присутствует препятствие;
  - 0 — отсутствует препятствие;
- D_f — показания датчика расстояния, направленного вперёд:
  - 1 — присутствует препятствие;
  - 0 — отсутствует препятствие;
- D_r — показания датчика расстояния, направленного вправо:
  - 1 — присутствует препятствие;
  - 0 — отсутствует препятствие.

Формат выходных данных

Одно число - количество секторов, которое возможно посетить. Если невозможно определить, то вывести число секторов посещённых первым роботом.

Примеры

Примеры входных данных и ответов к ним можно найти по ссылке `http://bit.ly/2LK5FrH`.

Решение

По условиям задачи нам известно количество роботов на поле, размер лабиринта по горизонтали и вертикали (в секторах), стартовые сектора каждого робота и направление каждого робота (рис. 4.41).

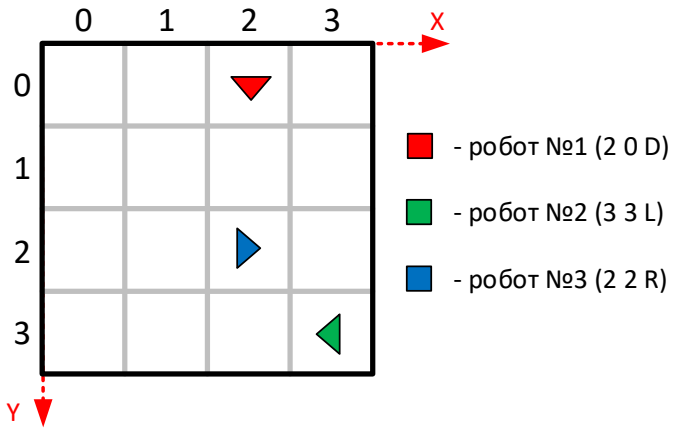


Рис. 4.41: Расположение роботов на старте

Представим лабиринт в виде неориентированного графа, где вершины графа - сектора лабиринта. В этом случае решение задачи заключается в составлении матрицы смежности графа, т.е. возможности проезда из любого сектора в смежные сектора. На старте матрица смежности "пустая". Нумерация вершин - абсолютная, начиная с "0"(рис. 4.42).

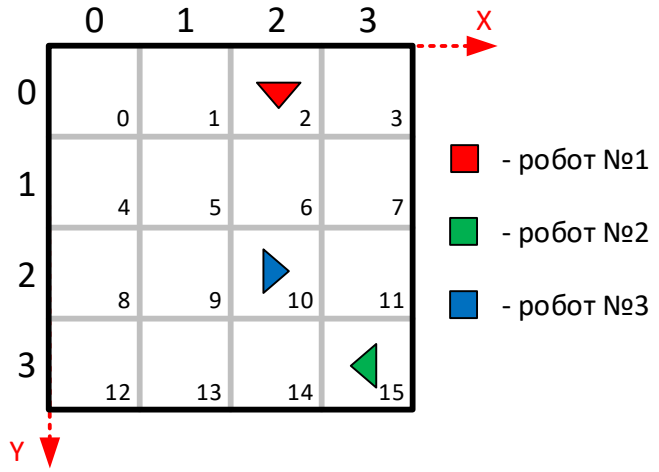


Рис. 4.42: Нумерация вершин (номер клетки = номер вершины графа)

Формула получения номера вершины из текущих координат робота (x, y):

$$cell\_number = y \cdot N_{cols} + x, \text{ где } N_{cols} \text{ - количество столбцов поля}$$

Например, чтобы получить номер стартовой клетки робота №1:

$$cell\_number = 0 \cdot 4 + 2 = 2$$

Для дискретной одометрии (информация о координатах робота и его направлении в текущий момент времени) для каждого робота нужно постоянно отслеживать его координаты секторов (x, y) и азимут (направление движения).

Далее по командам перемещения и показаниям датчиков для каждого робота заполняем матрицу смежности. По условиям задачи роботы перемещаются одновременно, т.е. за одну итерацию каждый робот совершает 1 действие (проезд прямо или поворот).

Разберем на примере 3 первые команды для каждого робота (таблица 4.4).

	Робот #1	Робот #2	Робот #3
Строка команд #1	F 1 1 0	F 1 0 1	F 0 1 1
Строка команд #2	R 1 0 0	F 1 1 0	L 0 0 1
Строка команд #3	F 0 0 1	R 1 0 0	F 1 0 1

Таблица 4.4: Примеры команд для перемещений роботов

Команда "F 1 1 0" обозначает, что робот проезжает прямо 1 сектор ("F"). Показания датчиков после остановки: слева - стена ("1"), впереди - стена("1"), справа - свободно ("0") и т.п.

Состояние лабиринта и положение роботов после выполнения 1 строки команд показаны на рис. 4.43.

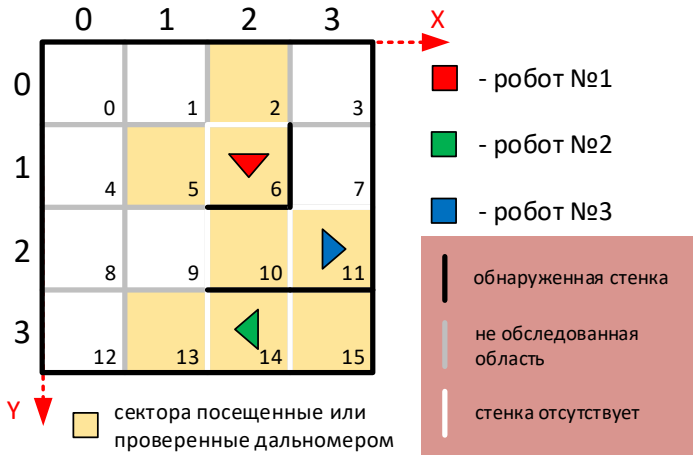


Рис. 4.43: Состояние лабиринта и положения роботов после 1й итерации

Робот #1, пример заполнения матрицы смежности.
Начальный сектор движения - 2, после проезда прямо (робот направлен вниз), робот оказывается в секторе 6. По показаниям датчиков корректируем матрицу смежности:

- Из сектора 2 можно проехать в сектор 6 ⇒ вершины 2 и 6 смежные
- Показания датчика слева = '1', т.е. между секторами 6 и 7 стена ⇒ вершины 6 и 7 не смежные.
- Показания датчика спереди = '1', т.е. между секторами 6 и 10 стена ⇒ вершины 6 и 10 не смежные.

- Показания датчика справа = ’0’, т.е. из сектора 6 можно проехать в сектор 5 $\Rightarrow$ вершины 6 и 5 смежные.

Подобным образом корректируем матрицу смежности после проезда каждого робота. Также необходимо учитывать посещенные сектора и/или проверенные дальномером.

Лабиринт и роботы после выполнения 2 строки команд:

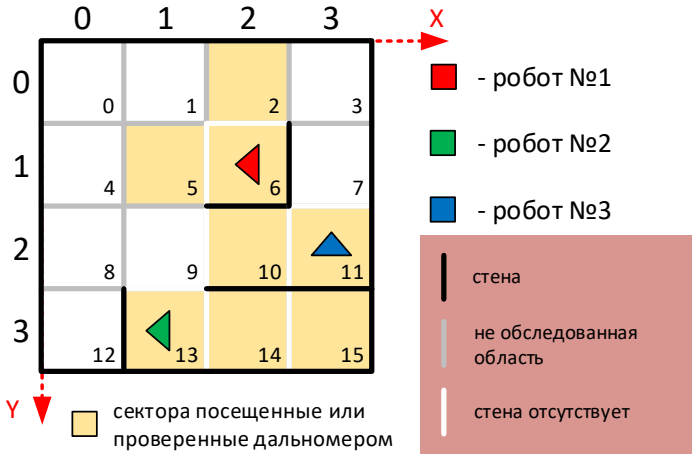


Рис. 4.44: Состояние лабиринта и положения роботов после 2й итерации

Лабиринт и роботы после выполнения 3 строки команд:

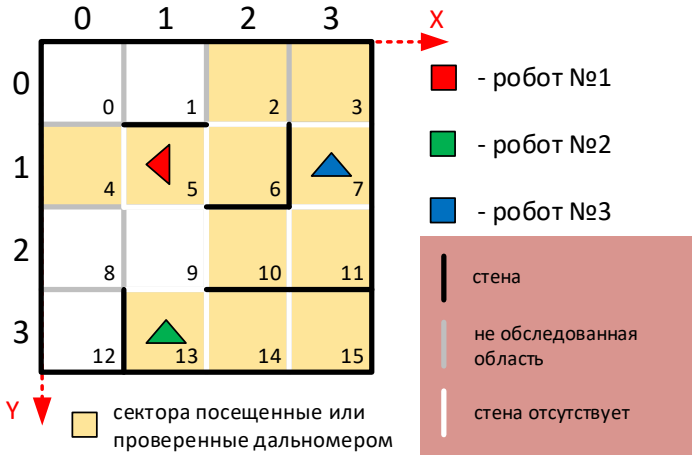


Рис. 4.45: Состояние лабиринта и положения роботов после 3й итерации

Лабиринт и роботы после выполнения всех команд (рис. 4.46).

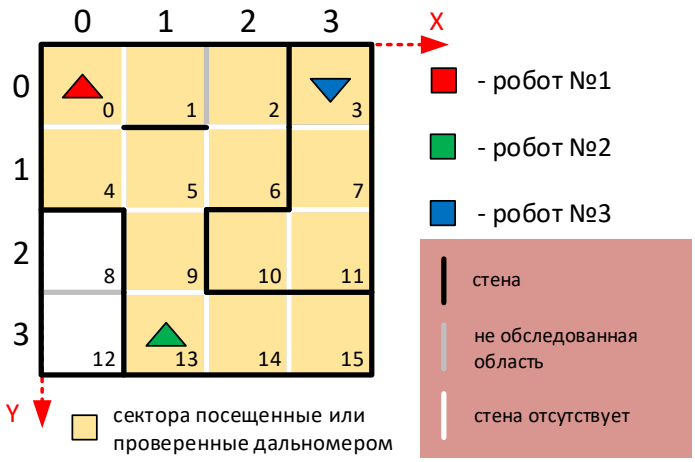


Рис. 4.46: Состояние лабиринта и положения роботов после выполнения всех итераций

После выполнения всех команд по перемещению роботов проверяем карту лабиринта на предмет наличия не посещенных и не проверенных дальномером секторов без явно обнаруженных стен, т.к. в этом случае мы не знаем точно - можно ли из этого сектора перемещаться в смежные сектора или нельзя, поэтому:

- Если такие сектора есть, то ответом будет количество секторов, которые проехал Робот #1
- Если таких секторов нет, то через алгоритм BFS проверяем количество секторов, которые сможет посетить Робот #1 начиная со стартовой позиции.
  В ответ выводим полученное количество секторов. В нашем примере это 10.

Ответ: 10 секторов

Пример программы-решения

Ниже представлено решение на языке Python3

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

def write_matrix(cell1, cell2, value):
    global matrix
    matrix[cell1][cell2] = int(not value)
    matrix[cell2][cell1] = int(not value)

def edit_matrix(m):
    global LMooved

    for rr in range(N):
        x, y, az = robots_moves[rr]
        cells = [0, 0, 0]                    # sensors: [left, forward, right]
        if az == 0:
            cells[0] = coord2cell(x - 1, y)
            cells[1] = coord2cell(x, y - 1)
            cells[2] = coord2cell(x + 1, y)
        elif az == 1:
            cells[0] = coord2cell(x, y - 1)
```

```
21              cells[1] = coord2cell(x + 1, y)
22              cells[2] = coord2cell(x, y + 1)
23          elif az == 2:
24              cells[0] = coord2cell(x + 1, y)
25              cells[1] = coord2cell(x, y + 1)
26              cells[2] = coord2cell(x - 1, y)
27          elif az == 3:
28              cells[0] = coord2cell(x, y + 1)
29              cells[1] = coord2cell(x - 1, y)
30              cells[2] = coord2cell(x, y - 1)
31
32          robot_obstacle = [False] * 3
33          for b in range(N):                          # проверяем каждого робота
34              for ss in range(3):                     # на пересечение с каждым дальномером
35                  if coord2cell(robots_moves[b][0], robots_moves[b][1]) == cells[ss]:
36                      robot_obstacle[ss] = True
37
38          cell = coord2cell(x, y)
39          for s in range(3):                           # 3 sensors
40              if cells[s] >= K * M or cells[s] < 0:
41                  continue
42
43              if not robot_obstacle[s]:
44                  write_matrix(cell, cells[s], measures[rr][m][s + 1])
45                  write_matrix(cells[s], cell, measures[rr][m][s + 1])
46
47              if not robot_obstacle[s] and not measures[rr][m][s + 1]:
48                  LMooved[cells[s]] = True
49
50  def cell2coord(cell):
51      y = int(cell / K)
52      x = cell - y * K
53      return (x, y)
54
55  def coord2cell(x, y):
56      # K - ширина лабиринта, секторов
57      return y * K + x
58
59  def bfs():
60      start = coord2cell(robots[0][0], robots[0][1])
61      visited = [False for i in range(K * M)]
62
63      path = []
64      queue = [start]
65      while len(queue) > 0:
66          p = queue.pop(0)
67          if not visited[p]:
68              visited[p] = True
69              path.append(p)
70
71              for i in range(K * M):
72                  if not visited[i] and matrix[p][i] > 0:
73                      queue.append(i)
74
75              x, y = cell2coord(p)
76              if x + 1 < K:
77                  if matrix[p][p + 1] < 0 and not LMooved[p + 1]:
78                      return []
79
80              if x - 1 >= 0:
```

```
                if matrix[p][p - 1] < 0 and not LMooved[p - 1]:
                    return []

            if y + 1 < M:
                if matrix[p][p + K] < 0 and not LMooved[p + K]:
                    return []

            if y - 1 >= 0:
                if matrix[p][p - K] < 0 and not LMooved[p - K]:
                    return []

    return path

data = sys.stdin.readlines()
N, K, M, cnt_sens = list(map(int, data[0].strip().split(' ')))
data.pop(0)

#N        - кол-во роботов на поле
#K        - ширина лабиринта, секторов
#M        - длина лабиринта, секторов
#cnt_sens - кол-во показаний датчиков

measures, robots = [], []
az = ['U','R','D','L']
for i in range(N):
    line = data[0].strip().split(' ')
    robots.append([int(line[0]), int(line[1]), az.index(line[2])])
    data.pop(0)

for move in range(N):
    measures.append([])
    for m in range(cnt_sens):
        line = data[move * cnt_sens + m].strip().split(' ')
        measures[-1].append([line[0]])
        for i in range(3):
            measures[-1][-1].append(int(line[i + 1]))

matrix = [[-1 for j in range(K*M)] for i in range(K*M)] # пустая матрица смежности
move_cells = [1 for i in range(N)] # подсчет посещенных клеток роботами
LMooved = [False for i in range(K * M)]        # просмотренные или посещенные клетки

robots_moves = []
for i in range(N):
    robots_moves.append(robots[i])                          # стартовые позиции роботов

for m in range(cnt_sens):
    for r in range(N):
        x, y, azimut = robots_moves[r]
        LMooved[coord2cell(x, y)] = True

        action = measures[r][m][0]
        if action == 'L':   azimut -= 1
        elif action == 'R': azimut += 1
        azimut %= 4

        if action == 'F':
            x0, y0 = x, y
            if   azimut == 0: y -= 1
            elif azimut == 1: x += 1
            elif azimut == 2: y += 1
```

```
141             elif azimut == 3: x -= 1
142             move_cells[r] += 1
143 
144             write_matrix(coord2cell(x,y),coord2cell(x0,y0), 0)
145 
146         robots_moves[r] = [x, y, azimut]
147 
148     edit_matrix(m)
149 
150 result = bfs()
151 if len(result) == 0 :
152     print (move_cells[0])
153 else:
154     print (len(result))
```

Задача 4.1.9. Планирование движения в лабиринте (15 баллов)

Роботы в количестве N, собранные по дифференциальной схеме, движутся по заранее известному лабиринту и представленному на рис. 4.47.

Необходимо доехать из начального в конечный сектор каждым роботом, если они движутся одновременно и параллельно и имеют следующие команды:

- F — проезд робота в следующий по ходу движения сектор;
- L — поворот робота в данном секторе налево и проезд в следующий по ходу движения сектор;
- R — поворот робота в данном секторе направо и проезд в следующий по ходу движения сектор.

Считать что роботы поворачиваются мгновенно, а после одновременно начинают движение вперёд с одинаковой скоростью.Робот является цилиндром и занимает $2/3$ площади клетки. Каждое перемещение он начинает и заканчивает в центре клетки.

Также гарантируется, что данных команд будет достаточно для выполнения задачи. При движении столкновение не допускается и перемещение необходимо осуществлять так, чтобы роботы получили наименьшее число команд. Иначе говоря, необходимо, чтобы сумма длин строк, содержащих команды передаваемые на робота в хронологическом порядке, без пробелов и запятых, была минимально возможной.

Формат входных данных

Первая строка содержит 1 целое число: N — количество роботов на поле ($1 \leqslant N \leqslant 3$).

Далее идет N строк, содержащие координаты старта и финиша каждого робота, а также направление робота при старте, т.е. одна строка имеет следующую структуру: x_s, y_s, dir, x_f, y_f, где:

- x_s — координаты старта данного робота по оси X;
- y_s — координаты старта данного робота по оси Y;
- dir — направление робота при старте:
  - U – робот направлен вверх;

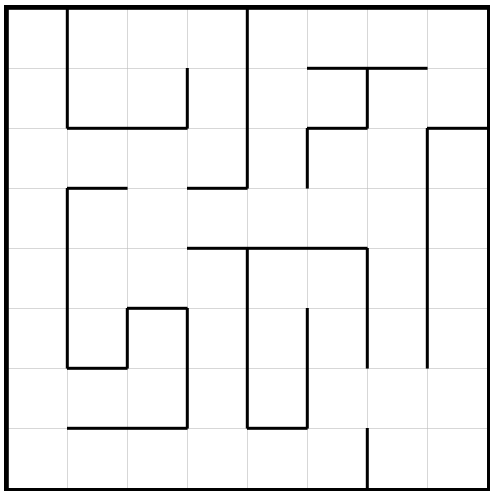


Рис. 4.47: Структура лабиринта для решения задачи

  - L – робот направлен влево;
  - D – робот направлен вниз;
  - R – робот направлен вправо;
- x_f — координаты финиша данного робота по оси X;
- y_f — координаты финиша данного робота по оси Y;

Все данные указаны через пробел, числа являются целыми.

Формат выходных данных

N строк, каждая строка содержит команды, передаваемые на данного робота в хронологическом порядке (самая первая команда расположена в начале строки), без пробелов и запятых.

Комментарии

Дополнительные наборы входных данных доступны по ссылке `http://bit.ly/2KtN4z9`.

Примеры

Пример №1

Стандартный ввод
2 6 7 U 0 2 7 7 U 1 2

Стандартный вывод
FLLRFFFFRFFFF FLRFFFLLRFFRL

Пример №2

Стандартный ввод
2 0 0 D 2 5 0 1 R 2 1
Стандартный вывод
FFFFFFLFL RFFFFFLFFLFFLRFRLFLL

Решение

Одним из вариантов решения данной задачи является составление графа, где вершины – состояние позиций роботов внутри лабиринта, а ребра - действия всех роботов, приведшие к переходу в это состояние.

Например, если по лабиринту перемещаются три робота, то ребро "FRF" определяет, что следующее состояние получается посредством перемещения первого робота прямо, второго - поворотом направо, третьего - также проездом прямо. Необходимо помнить, что согласно условиям задачи, поворот направо/налево - это поворот в текущем секторе направо/налево и проезд прямо.

При построении новых ребер необходимо учитывать, что роботы не должны сталкиваться. Также возможны такие ребра, где один из роботов стоит - т.е. в одном из предыдущих состояний он приехал в точку финиша.

На рис. 4.48 изображен пример графа для очень простого лабиринта и двух роботов. Очевидно, в таком графе можно применить поиск в ширину для нахождения вершины, когда все роботы находятся в требуемом секторе. Последовательность ребер из базового состояния в финишное состояние - будет кодировать путь каждого робота.

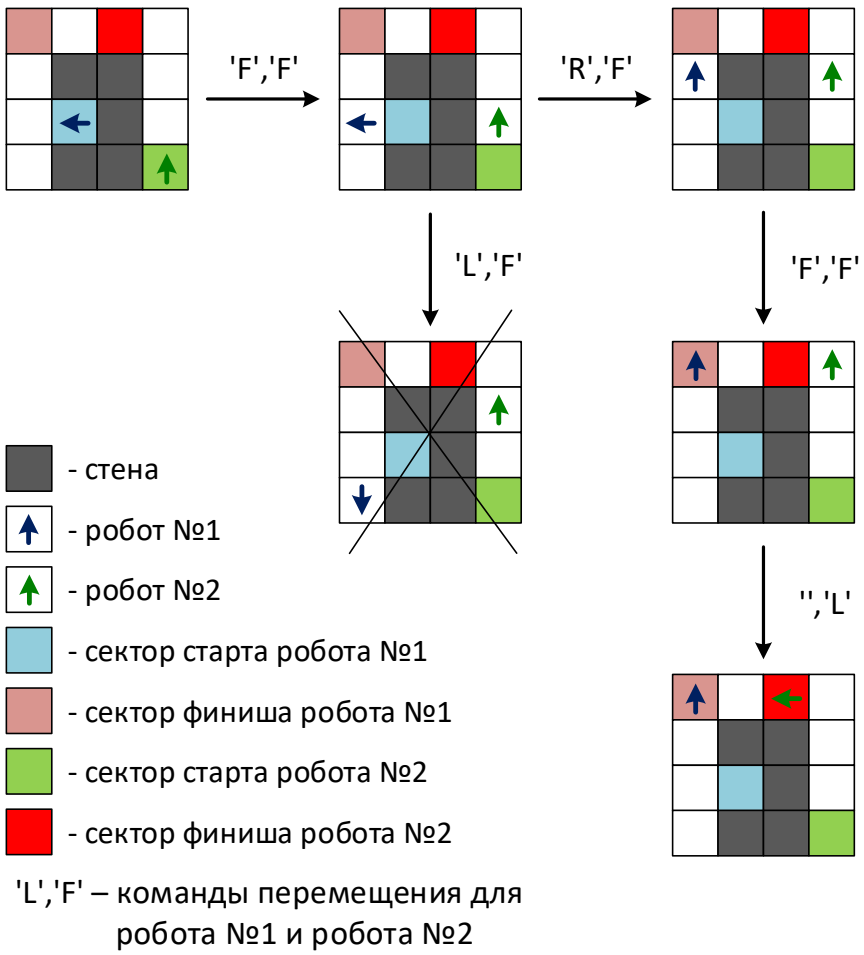


Рис. 4.48: Граф перемещения роботов

Пример программы-решения

Ниже представлено решение на языке C++

```
// Состояние робота - это пара чисел cell и dir робота
// Количество различных состояний робота для лабиринта размером AxB:
// K_robot = 4*A*B
// Состояние лабиринта - это упорядоченный набор состояний роботов
// Количество различных состояний робота для N роботов:
// K_maze = K_robot^N
// С ограничениями по времени и памяти программа справится с N <= 3

#include <iostream>
#include <vector>
#include <string>
#include <queue>
#include <algorithm>
#include <ctime>

using namespace std;

// Выводить ли отладочную информацию?
const bool debug = false;
```

```
20
21  // Размеры лабиринта
22  const int height = 8;
23  const int width = 8;
24
25  // Лабиринт
26  // adj[i] содержит массив вида {U, R, D, L}, где:
27  // U - номер клетки сверху от i-ой клетки
28  // R - номер клетки справа от i-ой клетки
29  // D - номер клетки снизу от i-ой клетки
30  // L - номер клетки слева от i-ой клетки
31  // Если с одной из сторон стена, то номер клетки = -1
32  const int adj[height * width][4] = {{-1, -1, 8, -1}, {-1, 2, 9, -1},
33      {-1, 3, 10, 1}, {-1, -1, 11, 2}, {-1, 5, 12, -1}, {-1, 6, -1, 4},
34      {-1, 7, -1, 5}, {-1, -1, 15, 6}, {0, -1, 16, -1}, {1, 10, -1, -1},
35      {2, -1, -1, 9}, {3, -1, 19, -1}, {4, 13, 20, -1}, {-1, -1, -1, 12},
36      {-1, 15, 22, -1}, {7, -1, -1, 14}, {8, 17, 24, -1},
37      {-1, 18, -1, 16}, {-1, 19, 26, 17}, {11, -1, -1, 18},
38      {12, -1, 28, -1}, {-1, 22, 29, -1}, {14, -1, 30, 21},
39      {-1, -1, 31, -1}, {16, -1, 32, -1}, {-1, 26, 33, -1},
40      {18, 27, 34, 25}, {-1, 28, -1, 26}, {20, 29, -1, 27},
41      {21, 30, -1, 28}, {22, -1, 38, 29}, {23, -1, 39, -1},
42      {24, -1, 40, -1}, {25, 34, 41, -1}, {26, 35, -1, 33},
43      {-1, -1, 43, 34}, {-1, 37, 44, -1}, {-1, -1, 45, 36},
44      {30, -1, 46, -1}, {31, -1, 47, -1}, {32, -1, 48, -1},
45      {33, -1, -1, -1}, {-1, -1, 50, -1}, {35, -1, 51, -1},
46      {36, -1, 52, -1}, {37, -1, 53, -1}, {38, -1, 54, -1},
47      {39, -1, 55, -1}, {40, 49, 56, -1}, {-1, 50, -1, 48},
48      {42, -1, -1, 49}, {43, -1, 59, -1}, {44, -1, -1, -1},
49      {45, 54, 61, -1}, {46, 55, 62, 53}, {47, -1, 63, 54},
50      {48, 57, -1, -1}, {-1, 58, -1, 56}, {-1, 59, -1, 57},
51      {51, 60, -1, 58}, {-1, 61, -1, 59}, {53, -1, -1, 60},
52      {54, 63, -1, -1}, {55, -1, -1, 62}};
53  // Количество роботов на поле
54  int robot_count;
55
56  // Номер клетки и направление старта роботов
57  vector <int> cell_start, dir_start;
58
59  // Номер клетки и направления финиша роботов
60  vector <int> cell_finish;
61
62  // Количество возможных состояний лабиринта
63  int state_count = 0;
64
65  // used[i] - просмотрено i-ое состояние лабиринта или нет
66  vector <bool> used;
67  // used[i] - из какого состояния получили i-ое состояние лабиринта
68  vector <int> from;
69  // actions[i] - какое действия совершили роботы, чтобы перейти в i-ое
70  // состояние лабиринта
71  // вместо
72  vector <char*> actions;
73
74  // path[i] - путь для i-ого робота
75  vector <string> path;
76
77  // Состояние лабиринта в которое перешли роботы, когда оказались на точках финиша
78  int state_finish = -1;
79
```

```
80  // Переводит символьное представление направления в числовое
81  // 'U' -> 0
82  // 'R' -> 1
83  // 'D' -> 2
84  // 'L' -> 3
85  int dir_to_int(char c) {
86      if (c == 'U')
87          return 0;
88      else if (c == 'R')
89          return 1;
90      else if (c == 'D')
91          return 2;
92      else if (c == 'L')
93          return 3;
94      else {
95          cerr << "Invalid direction!: " << c << "\n";
96          throw(1);
97      }
98  }
99
100 // Переводит вектор состоний робота в числовое представление
101 int state_to_int(vector <pair <int, int>> state) {
102     int res = 0;
103     for (auto el : state) {
104         res *= (4 * height * width);
105         res += el.first * 4 + el.second;
106     }
107     return res;
108 }
109
110 // Переводит числовое представление состояния лабиринта в вектор состояний робота
111 vector <pair <int, int>> state_to_vector(int state) {
112     vector <pair <int, int>> res;
113     while (res.size() != robot_count) {
114         res.emplace_back(state % (4 * height * width) / 4, state
115             % (4 * height * width) % 4);
116         state /= (4 * height * width);
117     }
118     reverse(res.begin(), res.end());
119     return res;
120 }
121
122 // Возвращает следующую комбинация для строки (например FFF -> FFL; FFR -> FLF;
123 // RRR -> END)
124 // (S - бездействовать)
125 string next_combination(string s) {
126     int i = s.size() - 1;
127     bool cont = true;
128     while (cont && i >= 0) {
129         cont = false;
130         if (s[i] == 'S')
131             s[i] = 'F';
132         else if (s[i] == 'F')
133             s[i] = 'L';
134         else if (s[i] == 'L')
135             s[i] = 'R';
136         else if (s[i] == 'R') {
137             s[i] = 'S';
138             cont = true;
139             --i;
```

```
140         }
141     }
142     if (i < 0)
143         return "END";
144     else
145         return s;
146 }
147 
148 // Считывание данных
149 void get_data() {
150     cin >> robot_count;
151     cell_start.resize(robot_count);
152     dir_start.resize(robot_count);
153     cell_finish.resize(robot_count);
154     state_count = 1;
155     for (int robot = 0; robot < robot_count; ++robot) {
156         int xs, ys, xf, yf;
157         char ds;
158         cin >> xs >> ys >> ds >> xf >> yf;
159         cell_start[robot] = xs + ys * width;
160         cell_finish[robot] = xf + yf * width;
161         dir_start[robot] = dir_to_int(ds);
162         state_count *= 4 * height * width;
163     }
164 }
165 
166 // Вывод входных данных
167 void print_data() {
168     cout << "--------------------\n";
169     for (int robot = 0; robot < robot_count; ++robot) {
170         cout << "Robot " << robot + 1 << endl;
171         cout << "start: " << cell_start[robot] << " " << dir_start[robot] << endl;
172         cout << "finish: " << cell_finish[robot] << endl;
173         cout << "--------------------\n";
174     }
175 };
176 
177 // Поиск в ширину по состояниям лабиринта
178 void bfs() {
179     if (debug)
180         cout << "bfs started\n";
181     used.resize(state_count, false);
182     from.resize(state_count, -1);
183     actions.resize(state_count);
184     vector <pair <int, int>> cur;
185     for (int robot = 0; robot < robot_count; ++robot)
186         cur.emplace_back(cell_start[robot], dir_start[robot]);
187     used[state_to_int(cur)] = true;
188     queue <int> q;
189     q.push(state_to_int(cur));
190     while (!q.empty()) {
191         int cur_st = q.front();
192         cur = state_to_vector(cur_st);
193         q.pop();
194         // Проверить, если все роботы на точках финиша
195         bool finish = true;
196         for (int robot = 0; robot < robot_count; ++robot)
197             if (cur[robot].first != cell_finish[robot]) {
198                 finish = false;
199                 break;
```

```
            }
        if (finish) {
            state_finish = state_to_int(cur);
            break;
        }
        // перебираем все строки действий вида FFF, FFL, FFR, ...
        string s = string(robot_count, 'S');
        while (s != "END") {
            auto next = cur;
            bool ok = true;
            for (int robot = 0; robot < robot_count; ++robot) {
                if (s[robot] == 'S') {
                    if (cur[robot].first == cell_finish[robot]) {
                        int next_dir = next[robot].second;
                        int next_cell = next[robot].first;
                    } else {
                        ok = false;
                        break;
                    }
                } else if (from[cur_st] == -1 || actions[cur_st][robot] != 'S') {
                    if (s[robot] == 'F'){
                        int next_dir = next[robot].second;
                        int next_cell = adj[next[robot].first][next_dir];
                        if (next_cell == -1) {
                            ok = false;
                            break;
                        }
                        next[robot].first = next_cell;
                        next[robot].second = next_dir;
                    } else if (s[robot] == 'L') {
                        int next_dir = (next[robot].second + 3) % 4;
                        int next_cell = adj[next[robot].first][next_dir];
                        if (next_cell == -1) {
                            ok = false;
                            break;
                        }
                        next[robot].first = next_cell;
                        next[robot].second = next_dir;
                    } else if (s[robot] == 'R') {
                        int next_dir = (next[robot].second + 1) % 4;
                        int next_cell = adj[next[robot].first][next_dir];
                        if (next_cell == -1) {
                            ok = false;
                            break;
                        }
                        next[robot].first = next_cell;
                        next[robot].second = next_dir;
                    }
                } else {
                    ok = false;
                    break;
                }
            }
            for (int i = 0; i < robot_count; ++i)
                for (int j = 0; j < robot_count; ++j) {
                    if (i != j && next[i].first == cur[j].first &&
                        next[i].second != next[j].second) {
                        ok = false;
                        break;
                    }
```

```
260                     if (i != j && next[i].first == next[j].first) {
261                         ok = false;
262                         break;
263                     }
264                 }
265             if (ok && !used[state_to_int(next)]) {
266                 used[state_to_int(next)] = true;
267                 int state_int = state_to_int(next);
268                 // convert string to C-string
269                 actions[state_int] = new char[robot_count + 1];
270                 for (int i = 0; i < robot_count; ++i)
271                     actions[state_int][i] = s[i];
272                 actions[state_int][robot_count] = '\n';
273                 from[state_to_int(next)] = state_to_int(cur);
274                 q.push(state_to_int(next));
275             }
276             s = next_combination(s);
277         }
278     }
279     if (debug)
280         cout << "bfs finished\n";
281 }
282 
283 void print_path() {
284     if (debug)
285         cout << state_finish << endl;
286     if (state_finish == -1) {
287         cout << "No solution!\n";
288     } else {
289         path.resize(robot_count, "");
290         int cur_state = state_finish;
291         while (from[cur_state] != -1) {
292             for (int robot = 0; robot < robot_count; ++robot)
293                 path[robot].push_back(actions[cur_state][robot]);
294             cur_state = from[cur_state];
295         }
296         for (int robot = 0; robot < robot_count; ++robot) {
297             reverse(path[robot].begin(), path[robot].end());
298             path[robot] += 'S';
299             path[robot] = path[robot].substr(0, path[robot].find('S'));
300             cout << path[robot] << endl;
301         }
302     }
303 }
304 
305 int main() {
306     get_data();
307     if (debug)
308         print_data();
309     unsigned int start_time = clock();
310     bfs();
311     print_path();
312     if (debug) {
313         unsigned int end_time = clock();
314         cout << "Execution time: " << (end_time - start_time) / 1000.0 << " s" << endl;
315     }
316     return 0;
317 }
```

Задача 4.1.10. Определение пересекающихся траекторий (10 баллов)

Имеется набор байт — трафик, собранный на концентраторе, во время общения в локальной сети нескольких устройств, включая робототехнические устройства.

Известно, что робототехнические устройства общались по протоколу, построенному поверх UDP (`https://ru.wikipedia.org/wiki/UDP`) и реализованному следующим образом: три целых числа – t_i, X_i, Y_i – каждое из которых занимает 4 байта, где t_i — время, в которое были сняты данные показатели координат, а X_i и Y_i — координаты местоположения робототехнической тележки в данный момент времени. При записи чисел использовалась нотация BigEndian (`https://en.wikipedia.org/wiki/Endianness`).

Необходимо определить IP-адреса (`https://ru.wikipedia.org/wiki/IPv4`) устройств, чьи траектории пересекались.

Считать, что между изменениями робототехнические тележки перемещались прямо. Гарантируется, что через данный концентратор проходит лишь необходимый трафик, т.е. отсутствуют пакеты, не относящиеся к данной задаче.

Формат входных данных

Первая строка содержит 1 целое число: N — количество переданых пакетов через концентратор ($4 \leqslant N \leqslant 10^3$).

Далее идут N строк, каждая из которых содержит один пакет, переданный через концентратор в побитовом формате, который содержит t_i, X_i, Y_i, где:

- t_i — время в мс, в которое были сняты данные показатели координат – 4 байта ($0 \leqslant t_i < 2^{32}$);
- X_i — координаты по оси X местоположения робототехнической тележки в данный момент времени — 4 байта ($0 \leqslant X_i < 2^{32}$);
- Y_i — координаты по оси Y местоположения робототехнической тележки в данный момент времени — 4 байта ($0 \leqslant Y_i < 2^{32}$).

Формат выходных данных

Необходимо вывести два IP-адресса в десятичном формате через пробел в порядке их возрастания – адреса устройств, чьи траектории пересекались.

В случае если пересекались несколько пар роботов, эти пары следуют в порядке первых пересечений траекторий робототехнических устройств.

В случае если таких пересечений нет, следует вывести -1.

Примеры

Примеры входных данных и ответов к ним можно найти по ссылке `http://bit.ly/2QkcNzv`.

Задача 4.1.11. Определение собственных координат (10 баллов)

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

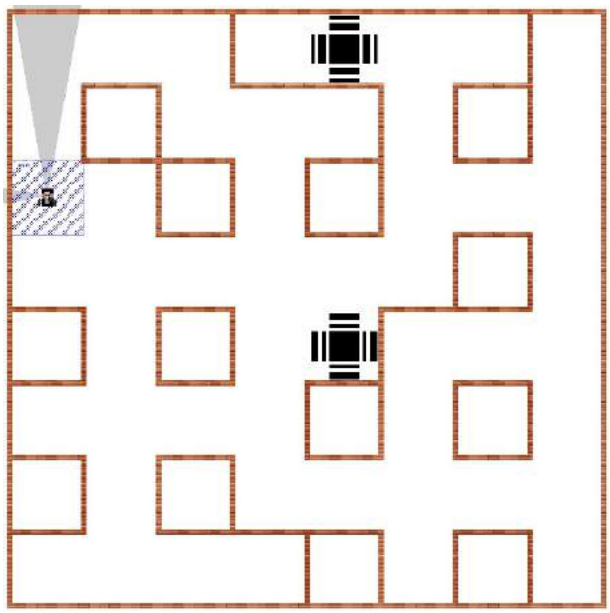


Рис. 4.49: Пример начального расположения робота

Для решения задачи робот запускается на поле, состоящим из 8x8 квадратных секторов. На поле между некоторыми секторами установлены препятствия для ограничения перемещения робота.

Роботу необходимо проехать некоторое количество секторов, заданное через входной файл *input.txt*, по правилу "левой руки", после остановиться и вывести на экран относительные координаты, т.е. свои координаты относительно точки старта, в формате (X, Y), где начало отсчета - точка старта, направление оси Y совпадает с первоначальным направлением движения, ось X направлена вправо перпендикулярно оси Y.

Конфигурация робота

Подключение моторов:

- Левый мотор - порт M3;
- Правый мотор - порт M4.

Подключение датчиков:

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

Формат входных данных

Входной файл содержит только одну строчку. В строке - целое число N ($5 \leqslant N \leqslant 40$), определяющее количество секторов, которое необходимо проехать. Сектор старта не учитывается в подсчёте.

Ограничения

Робот не должен выполнять задание дольше 3 минут.

Примеры

Для лабиринта, представленного на рис. 4.49, и при числе 7 во входных данных, робот должен остановиться в соответствии с рис. 4.50 и вывести на экран $(4, 1)$.

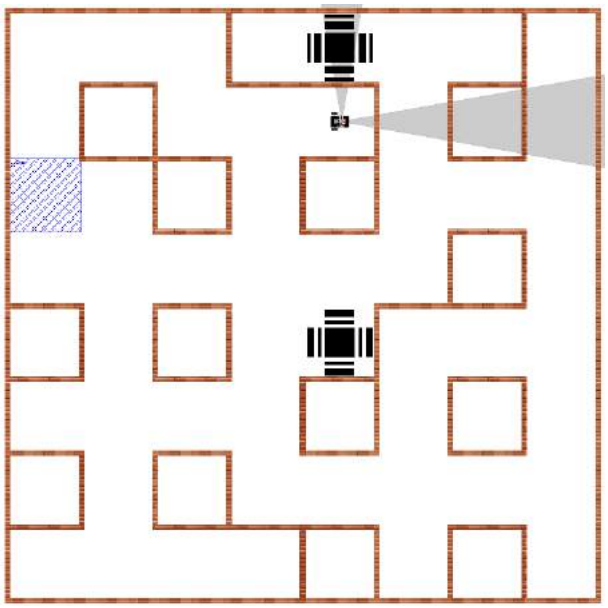


Рис. 4.50: Расположение робота, стартовавшего из позиции на рис. 4.49 и проехавшего 7 секторов

Решение

Для решения задачи в симуляторе TRIK-Studio определим следующую конфигурацию моторов и датчиков:

Оборудование	**Порт**
Правый мотор	M4
Левый мотор	M3
Датчик расстояния, направленный вперед	A1
Датчик расстояния, направленный влево	A2

Для перемещения в лабиринте нам необходимо подготовить функции:

- Проезд робота вперед на заданное расстояние
- Разворот робота на заданный угол (по энкодерам или гироскопу)
- Динамическое вычисление координат текущего сектора лабиринта (X; Y) и направления робота (азимут)
- Движение робота в лабиринте по правилу ”левой руки”

Алгоритмы ”прямолинейного движения на заданное расстояние” и ”разворот робота на месте” являются базовыми для начальной робототехники и подробно рассматривать мы их не будем.

Алгоритм ”левой руки”: Для движения по лабиринту по правилу ”левой руки” на роботе установлено 2 датчика расстояния: первый - спереди, второй – слева по ходу движения.

Псевдокод правила ”левой руки”:

1. *Если слева пусто (датчик возвращает расстояние больше, чем длина одного сегмента лабиринта), то:*
   (a) Поворот налево на 90 градусов
   (b) Проезд прямо на 1 сегмент
2. *иначе:*
   - если спереди пусто (датчик возвращает расстояние больше, чем длина одного сегмента), то:

   проезд прямо на 1 сегмент иначе: поворот направо на 90 градусов

Дискретная одометрия: Для вычисления азимута будем использовать следующее кодирование сторон света (см. рис.4.51)

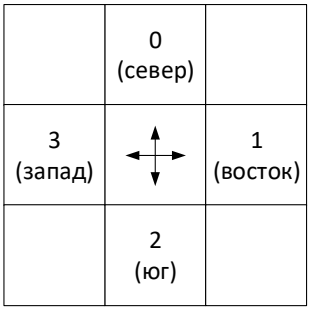


Рис. 4.51: Определение числовых вариантов сторон света

Вполне очевидно, что азимут робота меняется только при поворотах, а координаты сектора меняются только при перемещении между ними, но изменение координат зависит от азимута.

По условиям задачи робот в начальный момент времени установлен в направлении ”восток”. Полные координаты робота в момент старта $[x, y, azimut]$ показаны на рис. 4.52 :

Далее, согласно алгоритму ”левой руки”, робот проверяет возможность перемещения влево. В нашем случае - ”пусто”, т.е. робот поворачивает налево и проезжает вперед 1 сектор. При повороте налево меняется азимут с 1 на 0. При проезде прямо меняются координаты $(X; Y)$ в зависимости от азимута. Не забываем, что по условиям задачи робот на старте направлен вдоль оси Y, а ось X направлена вправо от робота (см. табл.4.5)

Подсчет посещенных секций делаем только при перемещении между ними (при поворотах секции не меняются).

Когда количество посещенных секций будет равно заданному значению - останавливаем робота и выводим на экран координаты робота. Обращайте внимание, в

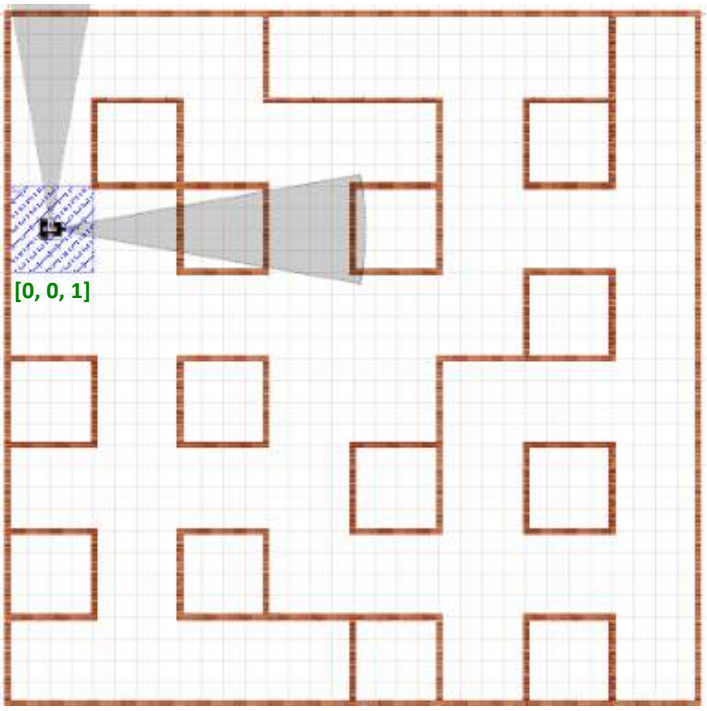


Рис. 4.52: Расположение робота на старте и начальные координаты робота [x, y, азимут]

Азимут			
0	1	2	3
X-1	Y+1	X+1	Y-1

Таблица 4.5: Пересчет координат робота (при движении "вперед"), в зависимости от текущего азимута

какое место экрана и в каком виде должен выводиться ответ.

Пример выполнения задания с количеством секций = 17 показан на рис.4.53.

Для нашего примера робот остановится в координатах $(-3; 2)$ с азимутом 3 (запад).

Ответ выводим на экран TRIK'а: **(-2, 3)**

Пример программы-решения

Ниже представлено решение на языке JavaScriot

```
function sign(num) { return num >= 0 ? 1 : -1}
function motors(mL, mR){ mR = mR || mL; brick.motor('M4').setPower(mL);
        brick.motor('M3').setPower(mR) }

function move(cm, turn){
        var L = (cm / (Math.PI * robot.D)) * robot.cpr
        L += encL()
        var sgn = sign(cm)
        motors(100 * sgn)
        if (sgn == 1){
```

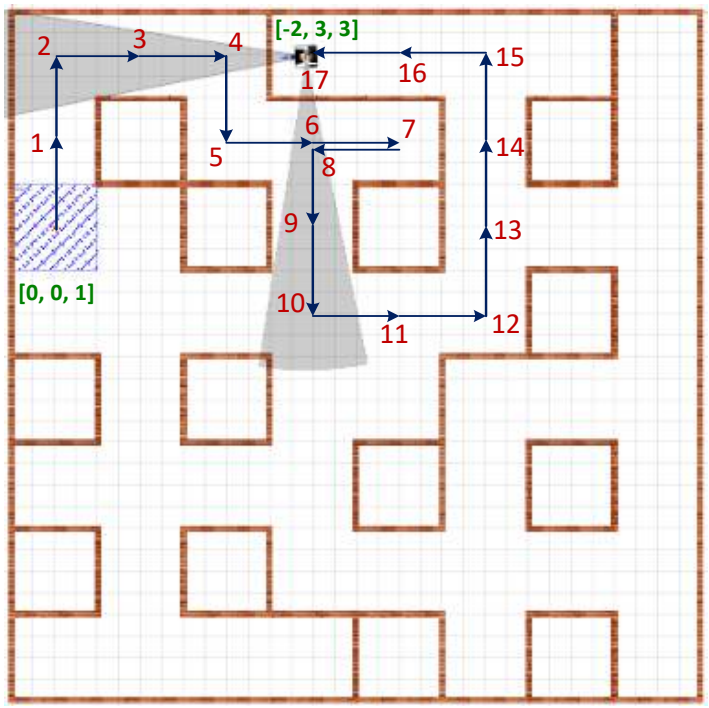


Рис. 4.53: Перемещения робота по правилу ”левой руки” с номерами посещенных секторов

```
                while (encL() <= L) script.wait(10)
        } else {
                while (encL() >= L) script.wait(10)
        }

        if (turn == undefined){
                step += 1
                switch (robot.a){
                        case 0: robot.x--; break;
                        case 1: robot.y++; break;
                        case 2:         robot.x++; break;
                        case 3:         robot.y--; break;
                }
        }
        motors(0)
}

robot = {
        D: 5.6,
        track: 17.5,
        x: 0,
        y: 0,
        a: 1,
        cpr: 360
}

cellLength = 17.5 * 4

sensF = brick.sensor('A1').read
sensL = brick.sensor('A2').read
encL  = brick.encoder('E4').read
encR  = brick.encoder('E3').read

```

```
44  wait = script.wait
45
46  steps = script.readAll('input.txt')
47  step = 0
48
49  function turn (angle){
50          var sgn = sign(angle)
51          move(17.5 * 0.5, true)
52          var lAngle = ((robot.track * angle) / (robot.D * 360)) * robot.cpr
53          lAngle += encL()
54
55          motors(50 * sgn, -50 * sgn)
56
57          if (sgn == 1){
58                  while (encL() <= lAngle) wait(10)
59                  robot.a++
60          } else {
61                  while (encL() >= lAngle) wait(10)
62                  robot.a--
63          }
64
65          if (robot.a > 3) robot.a = 0
66          if (robot.a < 0) robot.a = 3
67
68          motors(0)
69          move(17.5 * -0.5, true)
70  }
71
72
73  while (true){
74          if (sensL() > cellLength){
75                  turn(-90)
76                  move(cellLength)
77          } else {
78                  if (sensF() > cellLength)
79                          move(cellLength)
80                  else
81                          turn (90)
82          }
83          script.wait(100)
84          if (step == steps)
85                  break
86  }
87  motors(0)
88
89
90  // test
91
92  a = 1
93
94
95  out = '(' + robot.x + ',' + robot.y + ')'
96  brick.display().addLabel(out, 1, 1)
97  brick.display().redraw()
```