Второй отборочный этап

Второй отборочный этап проводится в командном формате в сети интернет, работы оцениваются автоматически средствами системы онлайн-тестирования. Продолжительность второго этапа составляет 54 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 114 баллов.

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

Задачи второго этапа

Задачи по информатике

Задача II.1.1.1. Планирование маршрута (10 баллов)

Плоский 2-звенный манипулятор, звенья которого соединены плоскими шарнирами с известным диапазоном вращения: $0 \leqslant q_{i,min} \leqslant q_i \leqslant q_{i,max} \leqslant 2\pi$, где $i \in 1, 2$, считая от основания манипулятора. Звенья манипулятора — тонкие стержни, круглые в сечении, диаметр стержней — 1 мм. Основание манипулятора зафиксировано в кординатах $(0, 0)$, диапазон работы шарниров и длина звеньев заданы через входные данные. Расположение и нумерация шарниров и звеньев манипулятора представлена на рисунке II.1.1.

Необходимо рассчитать конфигурацию манипулятора (углы звеньев), чтобы окончание манипулятора (рабочий орган), оказался в заданной через входные данные точке. Допускается точность в ± 0.01 рад. В случае если точка не может быть достигнута следует вывести -1.

Формат входных данных

Первая строчка содержит 4 вещественных числа через пробел — $q_{1,min}$, $q_{1,max}$, $q_{2,min}$, $q_{2,max}$, где:

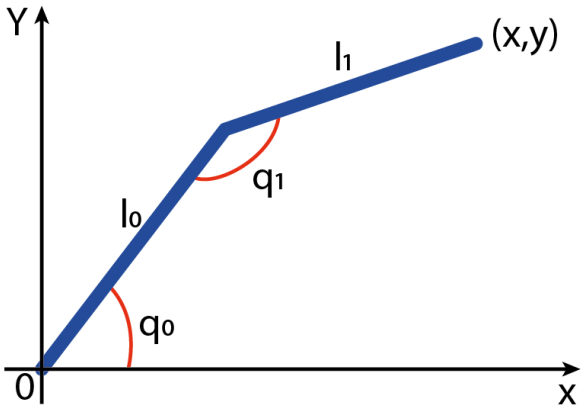


Рис. II.1.1: Расположение шарниров и звеньев манипулятора

- $q_{i,min}$ — минимальное значение i-того плоского шарнира в рад $(0 \leqslant q_{i,min} \leqslant 2\pi,\ \ i \in 1, 2)$;
- $q_{i,max}$ — максимальное значение i-того плоского шарнира в рад $(0 \leqslant q_{i,max} \leqslant 2\pi,\ \ i \in 1, 2)$;

Вторая строчка содержит два вещественных числа — $l_0,\ \ l_1$, где:

- l_i — длина i-того звена манипулятора, считая от основания, в мм $(0 < l_i \leqslant 10^6,\ \ i \in 1, 2)$.

Последняя строчка содержит два вещественных числа — $x,\ \ y$, где:

- x — значение координаты по оси X точки распложение рабочего органа в мм $(-10^6 \leqslant x \leqslant 10^6)$;
- y — значение координаты по оси Y точки распложение рабочего органа в мм $(-10^6 \leqslant y \leqslant 10^6)$.

Формат выходных данных

Одна строка, содержащая 2 вещественных числа через пробел — значения q_0 и q_1 последовательно с точностью в 0.01 рад. В случае если точка (x, y) не достижима следует вывести -1.

Примеры

Пример №1

Стандартный ввод

`4.427 5.736 1.996 3.628`
`16487.645 47356.757`
`25500.809 -58081.18`

Стандартный вывод
`5.32 2.88`

Пример №2

Стандартный ввод
3.063 5.263 2.507 3.069 30020.865 6565.169 -31732.301 -17849.818
Стандартный вывод
3.70 2.88

Решение

При известных конечных координатах рабочего органа и длинах звеньев необходимо найти соответствующие углы звеньев — называется это «инверсная (или обратная) кинематика».

Рассмотрим расчетную схему двухзвенного манипулятора на рис. II.1.2

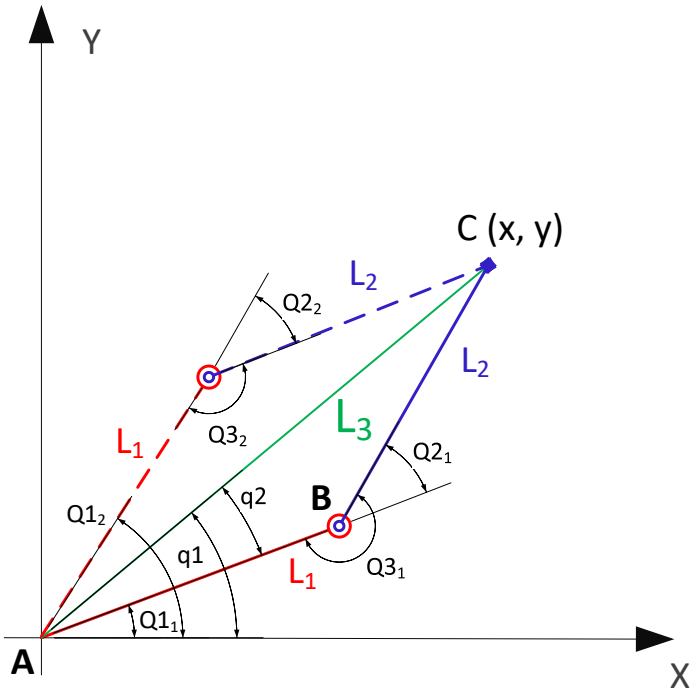


Рис. II.1.2: Расчетная схема манипулятора

где, известные величины по условиям задачи: L_1 — длина первого звена, см: 115.177 L_2 — длина второго звена, см: 122.680 x, y — координаты расположения рабочего органа, см: (13.459, 21.088) $(Q_{1_{min}}, Q_{1_{max}})$ — диапазон работы **первого** шарнира, рад: (4.649, 5.526) $(Q_{2_{min}}, Q_{2_{max}})$ — диапазон работы **второго** шарнира, рад: (5.0, 6.108)

расчетные величины: B — прямая, соединяющая начало координат с конечной точкой $q1$ — угол между прямой B и осью OX $q2$ — угол между прямой B и звеном L_1 $Q1_1, Q1_2$ — углы возможного расположения первого звена манипулятора, рад. Q_3 — угол поворота второго звена относительно первого звена манипулятора, рад.

В данной задаче необходимо найти подходящий вариант угла $Q1$ и найти угол $Q3$. Но с учетом ограничения рабочих углов шарниров, правильных решений может и не быть.

По схеме видно, что $Q1_1 = q1 - q2$ и $Q1_2 = q1 + q2$.

Сначала найдем L_3 по формуле II.1.1:

$$L_3 = \sqrt{x^2 + y^2} \Rightarrow \sqrt{13.459^2 + 21.088^2} = 25.01676 \tag{II.1.1}$$

Схема расположения звеньев манипулятора для нашего примера приведена на рис.II.1.3.

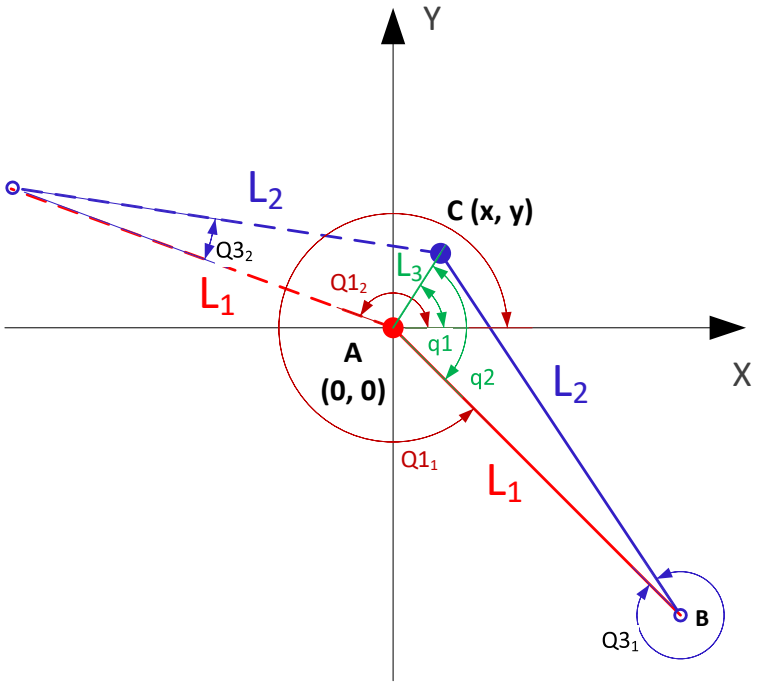


Рис. II.1.3: Расчетная схема манипулятора

Находим $q1$ по формулам II.1.2 и II.1.3:

$$q_1 = tan^{-1}\left(\frac{y}{x}\right) \Rightarrow tan^{-1}\left(\frac{21.088}{13.459}\right) = 1.00273 \tag{II.1.2}$$

$$q_1 = cos^{-1}\left(\frac{x}{B}\right) \Rightarrow cos^{-1}\left(\frac{13.459}{25.01676}\right) = 1.00273 \tag{II.1.3}$$

Находим $q2$ через теорему косинусов. Для плоского треугольника со сторонами (a, b, c) и углом α, противолежащим стороне a, справедливо соотношение:

$$a^2 = b^2 + c^2 - 2 \cdot b \cdot c \cdot cos(\alpha) \tag{II.1.4}$$

из теоремы косинусов для нашего примера:

$$\begin{gathered} {L_2}^2 = B^2 + {L_1}^2 - 2 \cdot B \cdot L_1 \cdot cos(q2) \quad \Rightarrow q2 = cos^{-1}\left(\frac{{L_1}^2 - {L_2}^2 + B^2}{2 \cdot B \cdot L_1}\right) \\ q2 = cos^{-1}\left(\frac{115.177^2 - 122.680^2 + 25.016^2}{2 \cdot 25.016 \cdot 115.177}\right) = 1.7732921 \end{gathered} \tag{II.1.5}$$

Теперь найдем варианты углов $Q1$ - $Q1_1$ и $Q1_2$:

$$\begin{gathered} Q1_1 = q1 - q2 \quad \Rightarrow \quad Q1_1 = 1.00273 - 1.7733 = -0.77056 \\ Q1_2 = q1 + q2 \quad \Rightarrow \quad Q1_1 = 1.00273 + 1.7733 = 2.77603 \end{gathered} \tag{II.1.6}$$

если значение получается отрицательным, то переводим его в положительный вариант:

$$Q1_1 = 2 \cdot \pi + Q1_1 = 5.513 \tag{II.1.7}$$

Опять же, по теореме косинусов найдем углы $Q2$. Как видно по рис. II.1.2, угол $Q2_1$ равен 180 — $\angle A$:

$$\begin{gathered} B^2 = L1^2 + L2^2 - 2 \cdot L1 \cdot L2 \cdot cos(\pi - Q2) \quad \Rightarrow \\ \angle B = cos^{-1}\left(\frac{L1^2 + L2^2 - B^2}{2 \cdot L1 \cdot L2}\right) = 0.2011 \end{gathered} \tag{II.1.8}$$

$$Q2_1 = \pi - \angle B \quad \Rightarrow Q2_1 = 2.940 \tag{II.1.9}$$

Угол $Q2_2$ равен обратному значению угла $Q2_1$, т. е. $Q2_2 = -Q2_1$.

Для вычисления внутренних углов $Q3_n$ определим, в какую сторону поворачивается точка C относительно точки B. Сделаем это через скалярное произведение векторов. Для этого нам достаточно вычислить координаты точки B_1 по формулам II.1.10 и II.1.11, а координаты точки C нам известны из условий задачи.

$$B_x = L_1 \cdot cos(Q1_1) = 115117.02 \cdot cos(5.513) = 82642.01 \tag{II.1.10}$$

$$B_y = L_1 \cdot sin(Q1_1) = 115117.02 \cdot sin(5.513) = -80225 \tag{II.1.11}$$

Находим скалярное произведение $\vec{b}(B_x, B_y) \cdot \vec{c}(C_x, C_y)$:

$$\vec{b} \cdot \vec{c} = \begin{vmatrix} B_x & B_y \\ C_x & C_y \end{vmatrix} = B_x \cdot C_y - B_y \cdot C_x = 2822483.14$$

В данном случае нам интересен только знак числа — если значение больше 0, значит поворот точки C относительно точки B будет против часовой стрелки, в противном случае - по часовой стрелке.

Исходя из этого условия вычисляем углы $Q3_1$ и $Q3_2$:

$$Q3_1 = 2 \cdot \pi - \angle B = 6.082 \tag{II.1.12}$$

Соответственно, второй возможный угол $Q3_2$ равен $\angle B$:

$$Q3_1 = \angle B = 0.2011 \tag{II.1.13}$$

В результате вычислений у нас есть две пары углов:

$$\begin{gathered} Q1_1 = 5.513, Q3_1 = 6.082 \\ Q1_2 = 2.776, Q3_2 = 0.201 \end{gathered}$$

Далее проверяем пары углов на вхождение в заданные диапазоны:
если $q1_{max} \geqslant Q1_1 \geqslant q1_{min}$ и $q2_{max} \geqslant Q3_1 \geqslant q2_{min}$, то в ответ выводим пару: $Q1_1 Q2_1$
если $q1_{max} \geqslant Q1_2 \geqslant q1_{min}$ и $q2_{max} \geqslant Q3_2 \geqslant q2_{min}$, то в ответ выводим пару: $Q1_2 Q2_2$

иначе выводим -1 — решений нет.

В нашем примере правильный ответ: 5.5136.082

Пример программы-решения

Ниже представлено решение на языке Python 3

```
from math import atan, acos, atan2, pi, sin, cos

def check(num):
    return num if num > 0 else pi_2 + num

q1_min, q1_max, q2_min, q2_max = map(float, input().split())
L1, L2 = map(float, input().split())
x, y = map(float, input().split())

L3 = (x*x + y*y) ** 0.5
pi_2 = 2 * pi
q1 = check(atan2(y,x))
q2 = acos((L1**2 - L2**2 + L3**2) / (2*L1*L3))

Q1_1 = check(q1-q2)
Q1_2 = check(q1+q2) % pi_2

angleB = acos((L1**2 + L2**2 - L3**2) / (2*L1*L2))
Q2_1 = pi - angleB
Q2_2 = -Q2_1

Bx, By = L1 * cos(Q1_1), L1 * sin(Q1_1)
Cx, Cy = Bx + L2 * cos(Q1_1 + Q2_1), By + L2 * sin(Q1_1 + Q2_1)

Q3_1 = pi_2 - angleB
Q3_2 = angleB

if Bx*Cy - By*Cx < 0:
    Q3_1 = angleB
    Q3_2 = pi_2 - angleB

Q1_1, Q1_2, Q2_1, Q2_2, Q3_1, Q3_2 = map(lambda x: round(x, 3), [Q1_1, Q1_2, Q2_1,
↪  Q2_2, Q3_1, Q3_2])

res = [-1]
if q1_max >= Q1_1 >= q1_min and q2_max >= Q3_1 >= q2_min:
   res = Q1_1,  Q3_1
elif q1_max >= Q1_2 >= q1_min and q2_max >= Q3_2 >= q2_min:
   res = Q1_2, Q3_2

print(*res)
```

Задача II.1.1.2. Планирование маршрута (8 баллов)

Дифференциальный робот расположен в известном лабиринте, структура которого передается через входные данные. Лабиринт представляет собой квадратный полигон (рис. II.1.4) размером $N \times N$ секторов с расположенными в нем препятствиями. Препятствия представляют собой объект, занимающий весь сектор целиком).

Необходимо проехать из начальной точки в конечную, если существуют следующие команды для робота:

- F — проезд роботов в следующий по направлению сектор;
- L — поворот робота на 90° налево внутри данного сектора;

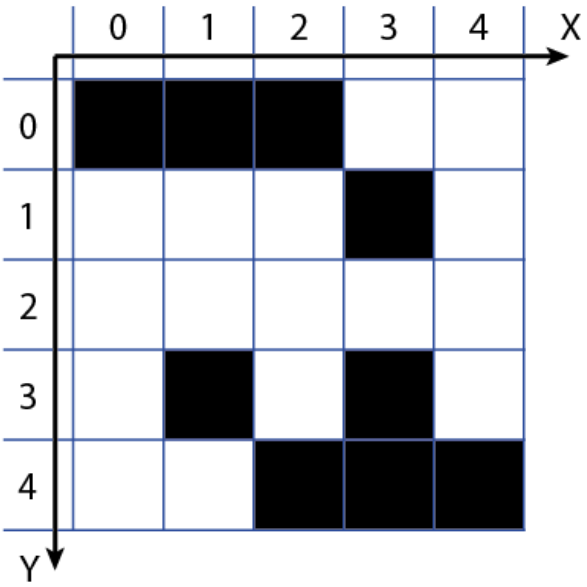


Рис. II.1.4: Пример тестового полигона

- R — поворот робота на 90° направо внутри данного сектора;

Гарантируется, что данный путь существует. Направление старта также известно и передается через входные данные, где:

- U — робот направлен вверх (в сторону отрицательного направления оси Y);
- L — робот направлен влево (в сторону отрицательного направления оси X);
- D — робот направлен вниз (в сторону положительного направления оси Y);
- R — робот направлен вправо (в сторону положительного направления оси X);

Формат входных данных

Первая строка содержит N, Dir, X_s, Y_s, X_f, Y_f — 1 целое число, символ и 4 целых чисел соответственно через пробел, где:

- N — длина лабиринта в секторах ($5 \leqslant N \leqslant 30$);
- Dir — направление старта робота;
- X_s — значение начальной координаты робота по оси X в секторах ($0 \leqslant X_s \leqslant N$);
- Y_s — значение начальной координаты робота по оси Y в секторах ($0 \leqslant Y_s \leqslant N$);
- X_f — значение конечной координаты робота по оси X в секторах ($0 \leqslant X_f \leqslant N$);
- Y_f — значение конечной координаты робота по оси Y в секторах ($0 \leqslant Y_f \leqslant N$);

Далее идет N строк. В каждой строке содержится N цифр через пробел, где:

- 0 — данный сектор свободен для посещения;
- 1 — в данном секторе расположено препятствие, т. е. данный сектор заблокирован.

Формат выходных данных

Одна строка, содержащая путь робота в виде последовательности команд без пробелов. Перемещение должно быть кратчайшим, т. е. полученная строка имеет

наименьшую длину.

Примеры

Примеры тестовых полигонов представлены по данной ссылке https://bit.ly/35T5IwJ.

Комментарии

Если правильных решений несколько, то принимается любое верное.

Решение

Решение задачи состоит в поиске оптимальных маршрутов перемещения робота из начального сектора в конечный сектор, с учетом направления робота на старте. Оптимальных маршрутов может быть несколько.

Оптимальным считается маршрут с минимальным количеством действий ("движение прямо"и "поворот— это отдельные действия), а не с минимальным количеством секторов перемещения.

Алгоритмы поиска оптимального пути описаны в презентации "Планирование и построение маршрута которая доступна на форме Университета Иннополис: .

Рассмотрим на примере перемещения робота в лабиринте 7x7 секторов. На рис. II.1.5 показаны 3 варианта перемещения робота из сектора "Старт"в сектор "Финиш".

Рис. II.1.5: Пример лабиринта и варианты перемещения робота

Количество действий для каждого варианта показаны на рис. II.1.6. Количество посещаемых секторов во всех 3 случаях одинаково — 8 секторов (включая секторы Старт и Финиш), но при движении по Варианту 3, робот совершает меньше поворотов, и, следовательно, меньше действий.

	Вперед	Поворот	Итого	Полный маршрут
Вариант 1	7	3	10	FFLFFRFLFF
Вариант 2	7	3	10	FFLFRFLFFF
Вариант 3	7	1	8	FFFLFFFF

Рис. II.1.6: Пример лабиринта и варианты перемещения робота

Пример программы-решения

Ниже представлено решение на языке Python 3

```
import math

def xy_to_cell(xx, yy):
    number = yy * map_size + xx
    return number if number >= 0 else -1

def generate_adj(pole):
    size = len(pole) * len(pole)
    matrix = [[0] * size for _ in range(size)]
    xx = [0, 1, 0, -1]
    yy = [-1, 0, 1, 0]

    for y in range(len(pole)):
        for x in range(len(pole)):
            cell = xy_to_cell(x, y) #y * len(pole) + x
            neighs = [(x + xx[i], y + yy[i]) for i in range(4)]
            #print(neighs)
            for X, Y in neighs:
                if X in range(0, map_size) and Y in range(0, map_size):
                    if pole[y][x] == 0 and pole[Y][X] == 0:
                        cell_neigh = xy_to_cell(X, Y)
                        matrix[cell][cell_neigh] = 1
                        matrix[cell_neigh][cell] = 1
    return matrix

def dijkstra(start, end, matrix = None):
    def build_path(pairs):
        prev = pairs[-1][1]
        path = [pairs[-1][0], prev]
        for p in range(len(pairs)):
            for w in range(len(pairs)):
                if pairs[w][0] == prev:
                  prev = pairs[w][1]
                  path.append(prev)
                  break
        return path[::-1]

    if matrix is None:
        matrix = global_map

    distances = [1000] * len(matrix)
    visited = [False] * len(matrix)
    distances[start] = 0
```

```
44      pairs, path = [], []
45      queue = [v for v in range(len(matrix[start])) if matrix[start][v] > 0]
46      queue.insert(0, start)
47      found = False
48
49      while queue:
50          min_weight = 1000
51          for w in queue:
52              if distances[w] < min_weight:
53                  min_weight, c = distances[w], w
54
55          for p in range(len(matrix)):
56              if matrix[c][p] > 0 and not visited[p]:
57                  if distances[p] > distances[c] + matrix[c][p]:
58                      distances[p] = distances[c] + matrix[c][p]
59                      pairs.append([p, c])
60
61                  if p == end:
62                      found = True
63                      break
64
65                  if queue.count(p) == 0:
66                      queue.append(p)
67
68          if found:
69              break
70
71          visited[c] = True
72          queue.pop(queue.index(c))
73
74      return build_path(pairs)
75
76
77  def build_actions(path, current_dir = None):
78      current_dir = current_dir or robot_dir
79
80      actions = []
81      variants = {
82                map_size:  ['LL','R','','L'],
83                1:  ['R','','L','LL'],
84                -1: ['L','LL','R',''],
85                -map_size: ['','L','LL','R']
86                }
87
88      for i in range(1, len(path)):
89          new_dir = path[i] - path[i-1]
90          actions.append(variants[new_dir][current_dir])
91          current_dir = variants[new_dir].index('')
92          actions.append('F')
93
94      return actions
95
96  def optimal(start, end, matrix = None):
97    if matrix is None:
98      matrix = global_map
99
100   az = robot_dir
101   cell = start
102   path = []
103
```

```
104   dops = {
105           -map_size: [0, 1, 2, 1],  # -> N
106            1: [1, 0, 1, 2],         # -> E
107            map_size: [2, 1, 0, 1],  # -> S
108           -1: [1, 2, 1, 0],         # -> W
109           }
110
111   while True:
112     neighs = [v for v in range(len(matrix[cell])) if matrix[cell][v] > 0 and v not in
      ↪  path]
113     path.append(cell)
114     # n-cell: -1 -> W, 1 -> E, 8 -> S, -8 -> N
115     lengths = {}
116     for n in neighs:
117       move = n - cell
118       az = list(dops.keys()).index(move)
119       p = dijkstra(n, end, matrix)
120       acts = ''.join(build_actions(p, az))
121       lengths[n] = len(acts) + dops[move][az]
122
123     cell_new = min(lengths, key = lengths.get)
124     az = list(dops.keys()).index(cell_new - cell)
125     cell = cell_new
126     neighs = [v for v in range(len(matrix[cell])) if matrix[cell][v] > 0]
127
128     if end in neighs:
129       path += [cell, end]
130       break
131
132   return path
133
134 ################ PROGRAM START ###################
135
136 out = []
137 data_in = input().split()
138 data_in[1] = 'URDL'.index(data_in[1])
139 map_size, robot_dir, x_start, y_start, x_finish, y_finish = map(int, data_in)
140
141 field = []
142 for _ in range(map_size):
143     field.append(list(map(int, input().split())))
144
145 global_map = generate_adj(field)
146 cell_start, cell_finish = xy_to_cell(x_start, y_start), xy_to_cell(x_finish, y_finish)
147 result = ''.join(build_actions(dijkstra(cell_start, cell_finish)))
148 print(result)
```

Задача II.1.1.3. Движение вдоль стены (12 баллов)

Робот, собранный по дифференциальной схеме оснащен одним ультразвуковым датчиком расстояния. Датчик расстояния установлен так, что показывает расстояние до препятствий, расположенных слева, относительно направления движения робота. Датчик (рис. II.1.7) установлен под углом 45° относительно направления движения робота.

Необходимо настроить движение робота таким образом, чтобы он перемещался вдоль стенки на расстоянии 200 мм. Известно, что робот начинает свое движение на расстоянии 700 мм в направлении вдоль препятствия. Гарантируется, что существу-

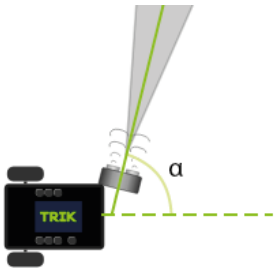


Рис. II.1.7: Расположение датчика на роботе

ет такая система управления (например, ПИД регулятор), позволяющая успешно проехать все тесты.

Рис. II.1.8: Пример теста

Конфигурация робота

Подключение моторов:

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

Подключение датчика:

- Датчик расстояния, направленный влево — порт D1

Примеры

Примеры тестовых полигонов представлены по данной ссылке https://bit.ly/3oF6J2a.

Комментарии

В задаче во всех тестах реальная физика, моторы и сенсоры **включены**.

Решение

По условиям задачи известно, что робот на старте находится на расстоянии 70 см от стены, а стена находится слева от робота по ходу движения.

Решением данной задачи является конфигурация и настройка программного регулятора для управления мощностями моторов. В качестве ошибки для регулятора принимается отклонение робота от стены.

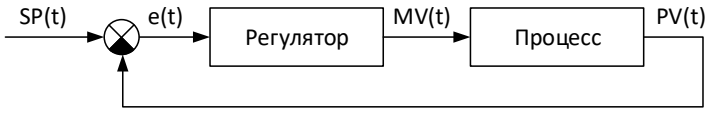
Возможный вариант решения: робот поворачивается налево на 90°, едет прямо до расстояния 25-30 от стены, поворачивает направо на 90° и двигается вдоль стены по регулированию через датчик расстояния.

В данном случае мы разберем другой вариант, когда робот двигается только по регулятору, начиная со старта.

В теории автоматического управления (ТАУ) термин "регулятор"известен как способ уменьшения вероятности ошибок и влиянию на последующую выходную характеристику.

Подробную информацию про варианты регулирования описано в статье на Википедии: `https://ru.wikipedia.org/wiki/ПИД-регулятор` (Статья про ПИД-регуляторы).

В общем виде система регулирования процессов показана на рис. II.1.9



где,

SP(t) – уставка (желаемое значение), Set Point

e(t) – ошибка управления (сигнал рассогласования), error

PV(t) – текущая величина процесса, Process Value

MV(t) – управляющие сигналы, Manipulated Value

Рис. II.1.9: Схема регулирования процесса

В зависимости от потребностей, регулятор может быть от самого простого — релейного, до ПИД-регулятора, где управляющее воздействие формируется из трех составляющих: П-пропорциональная, И-интегральая, Д-дифференциальная. В нашем случае мы рассмотрим вариант управления робота через ПИД-регулятор (рис. II.1.10)

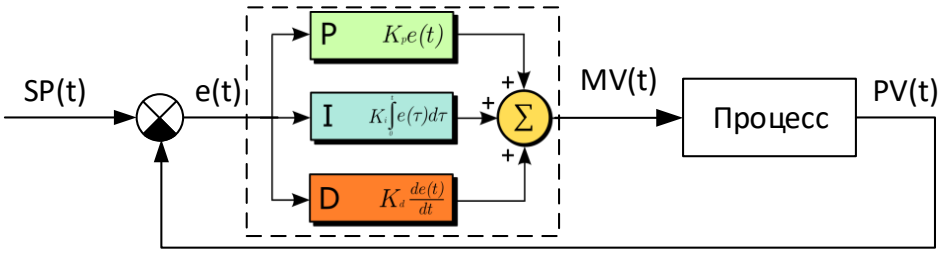


Рис. II.1.10: Схема регулирования через ПИД-регулятор

Найдем уставку для дальномера, т.к. по условиям задачи дальномер на роботе установлен под углом 45°, поэтому нам надо вычислить требуемое значение расстояния для УЗ-дальномера, при котором робот будет находиться на 20 см от стены. На схеме установки датчика (рис. II.1.11) видно, что требуемое расстояние робота до стены равно стороне $AB \angle ABC$. Диаграмма направленности УЗ-дальномера составляет 20°. Следовательно, дальномер “замечает” препятствия раньше в точке D, а не в точке C. Расстояние $BC = AB = 20$ см.

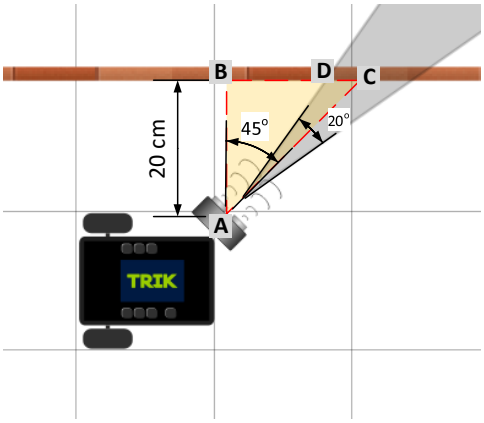


Рис. II.1.11: Геометрия установки дальномера

Найдем значение отрезка DC через пропорцию:

$$DC = \frac{10^\circ \cdot 20\text{ см}}{45^\circ} = 4\text{ см}$$

Соответственно, сторона $BD = BC - CD = 20 - 6 = 14\,\text{см}$. Вычисляем требуемое значение дальномера до стены по формуле II.1.1.3:

$$AD = \sqrt{AB^2 + BD^2} = \sqrt{20^2 + 16^2} = 25.6\text{ см}$$

Итак, в нашем случае $SP = 30\,\text{см}$, $e(t) = SP - PV$, где PV — показания дальномера, MV — величина изменения мощности моторов колес мобильного робота.

Пропорциональная составляющая

Формула расчета: $MV_P = e(t) \cdot K_p$

Начнем с подбора коэффициента пропорциональной составляющей регулятора. При значении $K_p > 1.5$ робот не успевает доехать до стены и начинает крутиться на месте, т.к. управляющее воздействие на моторы получается завышенным. При значении коэффициента $K_p < 1.5$ робот доезжает до стены на нужное расстояние 20 см, но не успевает выровняться вдоль стены и упирается в стену. Варианты показаны на рис. II.1.12.

Пока остановимся на $K_p = 1.3$, т.к. при нем робот доезжает до стены, хоть и не успевает выровняться.

Дифференциальная составляющая

Формула расчета: $MV_D = (e(t) - e(t-1)) \cdot K_d$, где $e(t-1)$ — предыдущее значение ошибки

Дифференциальная составляющая отражает скорость изменения ошибки, т. е. на чем большее значение изменилась ошибка по сравнению с предыдущей, тем больше будет значение MV_D. При незначительном изменении ошибки, дифференциальная составляющая будет практически равна 0.

По аналогии систем управления мобильными колесными роботами, зачастую $K_d = K_p \cdot 2..4$, отсюда примем $K_d = K_p \cdot 3 = 5.2$

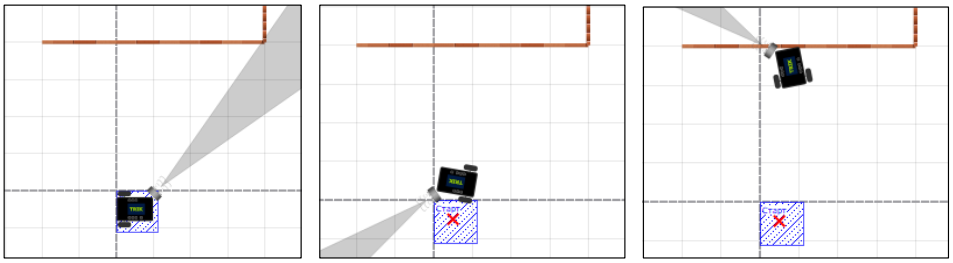


Рис. II.1.12: Положения робота при различных Kp

И уже с такими коэффициентами робот успешно проезжает вдоль стен до финишной секции.

Интегральная составляющая

Формула расчета: $MV_I = MV_I + e(t) \cdot K_i$ Интегральная составляющая является «накопителем» ошибки, т. е. чем больше система не может выровняться, тем больше интегральная составляющая. В нашем случае — когда робот на старте едет до первой стены или у стены крутой поворот (острый угол) и т. д.

Коэффициент интегральной составляющей очень маленький, по сравнению с остальными коэффициентами, примем его равным $K_i = 0.1$

Итоговое значение управляющего воздействия равно сумме всех составляющих: $MV = MV_P + MV_D + MV_I$

Для изменения мощности моторов будем применять MV с разными знаками, левый мотор: $+MV$, правый мотор: $-MV$.

Пример программы-решения

Ниже представлено решение на языке Python 3

```
import math

def motors(power_l, power_r = None):
    power_r = power_l if power_r is None else power_r
    motor_l(power_l)
    motor_r(power_r)


################ PROGRAM START ###################

motor_l = brick.motor('M4').setPower
motor_r = brick.motor('M3').setPower
sens_l = brick.sensor('D1').read
power_max = 80

wait = script.wait
pi = 3.141592

distance_to_wall = 30
```

```
20
21  Kp, Ki, Kd = 1.3, 0.1, 5.2
22
23  err = [0] * 10
24  while True:
25    err.append(distance_to_wall - sens_l())
26    uP = err[-1] * Kp
27    uI = (sum(err)/len(err)) * Ki
28    uD = (err[-1] - err[-2]) * Kd
29    u = uP + uI + uD
30    err = err[1:]
31
32
33
34    motors(power_max + u, power_max - u)
35    wait(30)
```

Задача II.1.1.4. Перемещение в лабиринте (16 баллов)

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

Необходимо в заранее неизвестном полигоне размером 8×8 секторов найти наименьшее расстояние между **ближайшей** тумбой к границе полигона и этой границей. Сектор — квадрат шириной 700 мм. Тумба занимает весь сектор и располагается так, что занимает либо весь сектор, либо со смещением в пол сектора. На полигоне может располагаться N $(1 \leqslant N \leqslant 6)$ тумб.

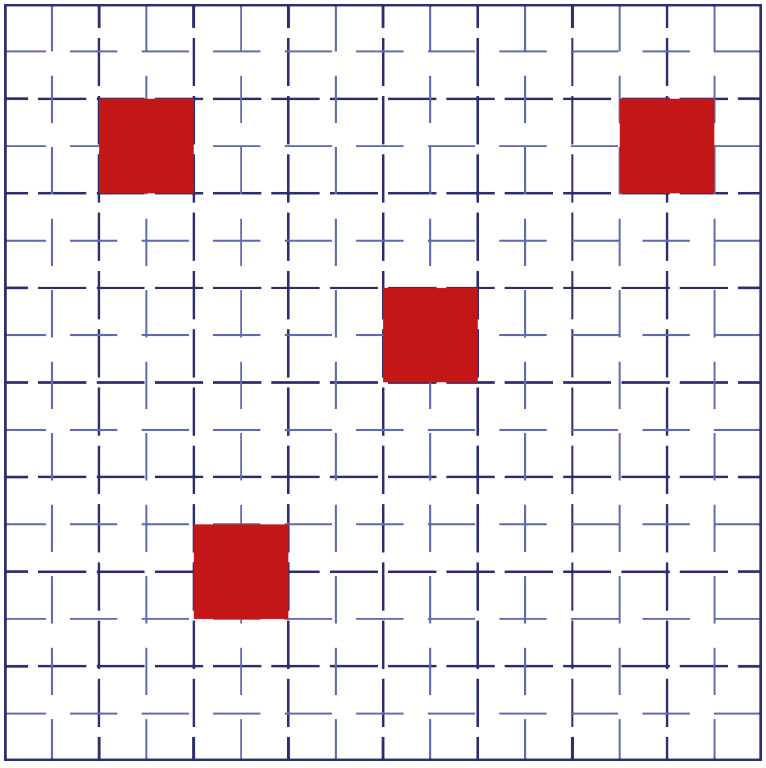


Рис. II.1.13: Пример расположения тумб на полигоне

Формат выходных данных

Определить расстояние между границей полигона и ближайшей тумбой. Необходимо вывести это значение на экран в мм с округлением вниз до целых чисел.

Конфигурация робота

Подключение моторов:

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

Подключение датчиков:

- Датчик расстояния, направленный вперед — порт D1.
- Датчик расстояния, направленный влево — - порт A1.
- Датчик расстояния, направленный вправо — порт A2.
- Датчик расстояния, направленный назад — порт D2.

Примеры

Примеры тестовых полигонов представлены по данной ссылке https://bit.ly/2Wcc84o.

Комментарии

В задаче во всех тестах реальная физика, моторы и сенсоры **выключены**.

Решение

Декомпозиция задачи:

1. Определить сторону расположения стены полигона относительно робота
2. Подъехать к стене
3. Начать движение вдоль стены до первого угла полигона
4. Вычислить координаты начального сектора
5. Продолжить движение вдоль стены по периметру полигона до начального сектора
6. В каждом секторе поворачиваться направо на 90 градусов и замерять расстояния дальномером
7. По приезду в начальный сектор вывести минимальное расстояние от стены до тумбы, с учетом геометрии установки дальномера на роботе

По условиям задачи на виртуальной модели робота в среде TRIK Studio установлены дальномеры двух типов — ИК-дальномер (слева и справа по ходу движения робота) с максимальной дальностью измерения 80 см и УЗ-дальномер (спереди и сзади) с максимальной дальностью измерения — 300 см (в TRIK Studio).

Для определения стороны расположения стены полигона, «просканируем» пространство возле робота всеми дальномерами: 4 дальномера — по 90° на каждый (см.

II.1.14). Спереди и сзади ограничимся расстоянием 280 см, справа и слева — 80 см, одновременно подсчитывая количество замеров в пределах указанных диапазонов (препятствия). На стороне стены количество замеров получается максимальным.

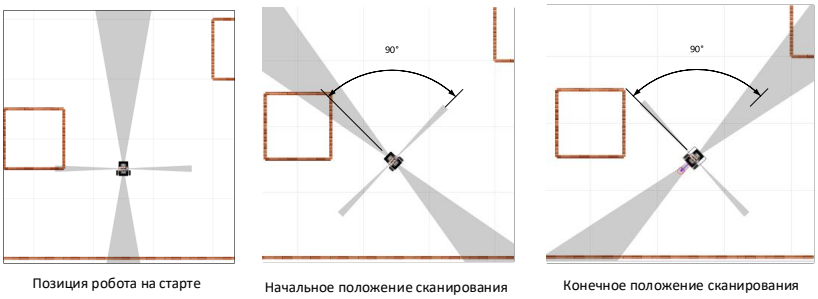


Рис. II.1.14: Сканирование пространства на старте

Разворачиваем робота в сторону стены и едем до расстояния 17 см от стены (по условиям задачи тумбы — это квадрат со стороной 70 мм, но они могут размещаться или целиком в секторе полигона или со смещением на половину сектора, отсюда примем минимально возможное расстояние для движения робота вдоль стены — 35 см и робот доезжает до середины этого расстояния). Робот стартует всегда в центре сектора, значит по оси, совпадающей со стеной, робот тоже находится в центре сектора. У стены поворачиваем на 90° направо (объезд полигона совершаем по часовой стрелке).

Пример показан на рис.II.1.15 (здесь и далее на примерах цветными линиями отмечается путь движения робота)

После каждого перемещения на расстояние равное размеру сектора, поворачиваемся роботом направо на 90° (т.к. установленные по бокам ИК-дальномеры замеряют не более чем на 80 см), замеряем расстояние, если оно меньше минимального, измеренного ранее — запоминаем, и одновременно пересчитываем координаты робота (X, Y, азимут). Угол полигона определяем по дальномеру спереди. Пересчет координат робота необходим для определения завершения сканирования периметра полигона.

Правильный ответ для данного полигона — 70 см. Полный путь движения робота и расстояния показаны на рис.II.1.16.

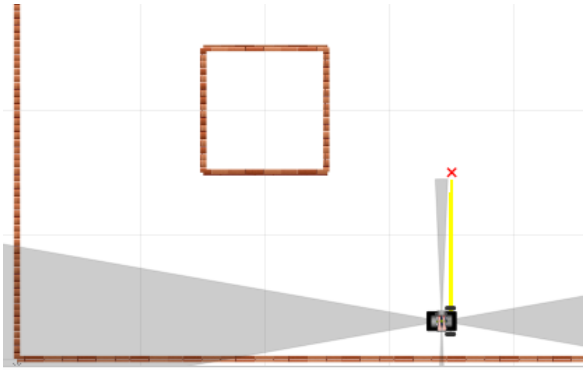


Рис. II.1.15: Робот в начале движения вдоль стены полигона

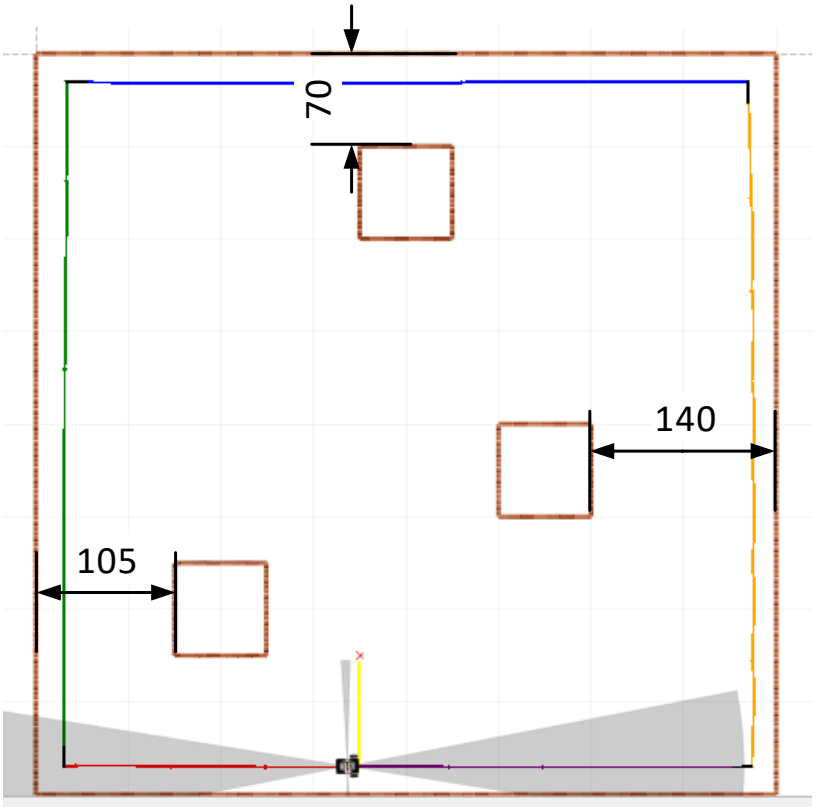


Рис. II.1.16: Результат выполнения задания. Ответ — 70 см.

Пример программы-решения

Ниже представлено решение на языке Python 3

```
import math

class Robot:
  def __init__(self, D = 5.6, track=15.4, motor_left = 4, motor_right = 3, x = 0, y =
  ↪  0, az = 0):
    self.D = D
    self.track = track
    self.motor_left = motor_left
    self.motor_right = motor_right
    self.x = x
    self.y = y
    self.az = az


class Motors:
  def __init__(self, left=4, right=3, cpr = 360, p_min = 40, p_max = 100):
    self.mL = brick.motor(f"M{left}").setPower
    self.mR = brick.motor(f"M{right}").setPower
    self.eL = brick.encoder(f"E{left}").read
    self.eR = brick.encoder(f"E{right}").read
    self.cpr = 360
    self.p_min = p_min
    self.p_max = p_max


  def power(self, p_l, p_r = None):
```

```
26      if p_r is None:
27        p_r = p_l
28      self.mL(p_l)
29      self.mR(p_r)
30
31    def encoders(self, avg = False):
32      return (self.eL(), self.eR()) if not avg else (self.eL() + self.eR())/2
33
34
35  class Gyro:
36    def __init__(self):
37      self.angles = [0, 90, 180, -90]
38      brick.gyroscope().calibrate(1000)
39      script.wait(1100)
40
41    def yaw(self):
42      return brick.gyroscope().read()[6]
43
44    def turn(self, side, v_turn = 90):
45      robot.az += side
46      robot.az %= 4
47
48      angle = self.angles[robot.az]
49      delta = angle % 360 - self.yaw()/1000 % 360
50      sgn = sign(delta)
51      sgn *= -1 if abs(delta) > 180 else 1
52
53      motors.power(v_turn * sgn, -v_turn * sgn)
54      while abs(angle - self.yaw()/1000) >= 7:
55        wait(20)
56      motors.power(0)
57
58
59    def move(self, path = 1000, wall = 15, dist = 0, by_wall = True):
60      gyro0 = self.angles[robot.az] * 1000
61      path = cm2cpr(path) + motors.eL()
62
63      while motors.eL() < path and sens_f() > wall:
64        y = self.yaw()
65        u = (gyro0 - y) / 1000 * 1.2
66        if gyro0 == 180000:
67          u = ((gyro0 - abs(y)) * sign(y))/1000 * 0.9 + ((18 - sens_l()) * 1.5 if
            ↪  by_wall else 0)
68
69        motors.power(motors.p_max + u, motors.p_max - u)
70        script.wait(30)
71
72        if sens_r() < dist:
73          dist = sens_r()
74
75      motors.power(0)
76
77      robot.x += neighs[robot.az][0]
78      robot.y += neighs[robot.az][1]
79
80      return dist
81
82
83  def sign(num):
84    return 1 if num >= 0 else -1
```

```
85
86
87  def cm2cpr(cm):
88    return cm / (robot.D * pi) * motors.cpr
89
90  def xy_to_cell(x, y, width = 8):
91    return y * width + x
92
93
94  def turn_enc(angle, v = 50):
95    angle *= 1
96    sgn = 1 if angle >= 0 else -1
97    path = robot.track * angle * motors.cpr / (robot.D * 360)
98    path += motors.eL()
99
100   motors.power(v * sgn, -v * sgn)
101   while motors.eL() * sgn < path * sgn:
102     script.wait(10)
103   motors.power(0)
104
105   robot.az += sgn
106   robot.az = robot.az % 4
107
108
109 def move_enc(cm):
110   global robot
111   k = 1
112   path_end = cm2cpr(cm) * k + motors.encoders(True)
113
114   err_old = 0
115   eL0, eR0 = motors.encoders()
116   while motors.encoders(True) <= path_end:
117     err = (motors.eL() - eL0) - (motors.eR() - eR0)
118     u = err * 0.5 + (err - err_old) * 1.7
119     err_old = err
120     motors.power(motors.p_max - u, motors.p_max +u)
121     script.wait(10)
122   motors.power(0)
123   robot.x += neighs[robot.az][0]
124   robot.y += neighs[robot.az][1]
125
126 def print_display(text):
127   brick.display().addLabel(text, 0, 0)
128   brick.display().redraw()
129
130 def goto_wall():
131   az = robot.az
132   turn_enc(-45)
133
134   motors.power(50,-50)
135   sensors = [[],[],[],[]]
136   while motors.eL() < 125:
137     sensors[0].append(sens_f() < 280)
138     sensors[1].append(sens_r() < 80)
139     sensors[2].append(sens_b() < 280)
140     sensors[3].append(sens_l() < 80)
141     wait(10)
142   motors.power(0)
143
144   turn_enc(-45)
```

```
145   robot.az = az
146
147   sides = [sum(i) for i in sensors]
148   max_side = sides.index(max(sides))
149   for _ in range(max_side):
150     gyro.turn(1)
151
152   brick.marker().down('yellow')
153   gyro.move(by_wall = False)
154   gyro.turn(1)
155
156
157 ################ PROGRAM START ##################
158 robot = Robot()
159 motors = Motors(robot.motor_left, robot.motor_right)
160 gyro = Gyro()
161
162 sens_l = brick.sensor('A1').read
163 sens_f = brick.sensor('D1').read
164 sens_r = brick.sensor('A2').read
165 sens_b = brick.sensor('D2').read
166
167 cell_length = 70
168 wait = script.wait
169 pi = 3.141592
170 neighs = [(0, -1), (1, 0), (0, 1), (-1, 0)]
171
172 goto_wall()
173
174 distances = []
175 blank_cells = 0
176 start_cells = 1
177 end_cell = 6
178 clrs = ['red', 'green', 'blue', 'orange', 'purple', 'brown', 'magenta']
179 result = -1
180 max_ir_distance = 79
181 max_us_distance = 230
182
183 for side in range(5):
184   if side > 0:
185     #brick.marker().down('black')
186     gyro.move(path = 16)
187
188   if side == 4:
189     end_cell = 8 - start_cells
190
191   #brick.marker().down(clrs[side])
192   gyro.move(cell_length * blank_cells)
193
194   blanks = []
195
196   for move in range(blank_cells, end_cell):
197     gyro.move(path = cell_length)
198     start_cells += 1 if side == 0 else 0
199
200     if sens_r() > max_ir_distance:
201       to_wall = sens_f()
202
203       gyro.turn(1)
204       distances.append(int(sens_f() + sens_b()))
```

```
205       blanks.append((sens_f() if sens_f() < max_us_distance else max_us_distance) //
          ↪  cell_length)
206       gyro.turn(-1)
207
208     else:
209       if sens_r() < 20:
210         result = 35
211         break
212
213       else:
214         distances.append(int(sens_r() + sens_l()))
215         blanks.append((sens_f() if sens_f() < max_us_distance else max_us_distance) //
            ↪  cell_length)
216
217     if side == 0 and sens_f() < cell_length:
218       break
219
220
221   if side == 0:
222
223     blanks.pop(-1)
224
225
226   if len(blanks) > 2:
227     blank_cells = min(blanks[-3:])
228   else:
229     blank_cells = len(blanks)
230
231   if side == 4 or result > -1:
232     break
233
234   gyro.move()
235   gyro.turn(1)
236
237
238 result = min(distances) if result == -1 else result
239 print_display(result)
240 print(result)
```

Задача II.1.1.5. Управление заднеприводной тележкой (16 баллов)

Робот, собранный по типу заднеприводной тележки с поворачивающейся передней осью (заднеприводной автомобиль) перемещается по полигону.

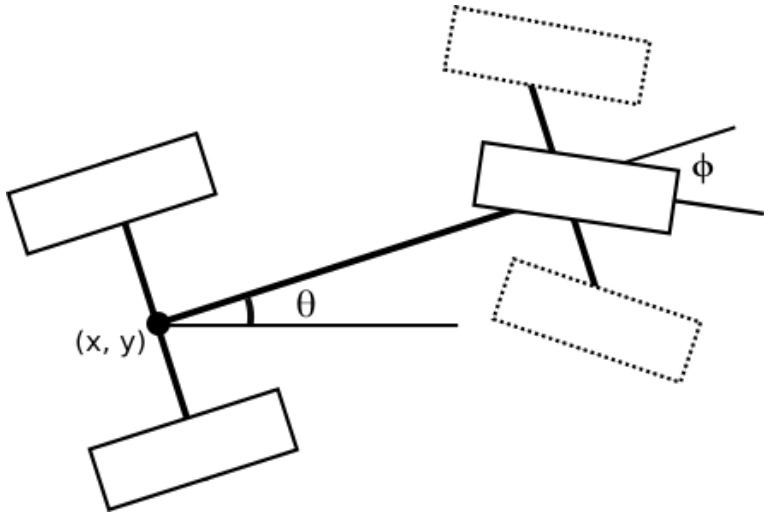


Рис. II.1.17: Пример заднеприводной тележки

fig:carExample

В процессе движения известны показания угловой скорости (ω в рад/с) поворота переднего колеса и линейной скорости (v в м/с) полученные через равные промежутки времени dt(в с). Необходимо найти координаты расположения центра задней оси через некоторое время. Расстояние между передней и задней осью – 2.7 м. Угловая и линейная скорость достигается мгновенно в момент получения данных показателей, т. е. через каждые dt. Гарантируется, что в конце робот остановился ($v = 0$ м/с и $\omega = 0$ рад/с).

Формат входных данных

Первая строчка содержит 4 вещественных чисел и 2 целых числа соответственно через пробел: $X,\ Y,\ \theta,\ \phi,\ dt,\ N$, где:

- X — координата по оси X начального расположения центра задней оси в м ($-10^6 < X < 10^6$);
- Y — координата по оси Y начального расположения центра задней оси в м ($-10^6 < X < 10^6$);
- θ — угол между осью тележки и положительным направлением оси X в рад ($-\pi \leqslant \theta < \pi$);
- ϕ — угол поворота колес относительно оси тележки в рад ($-\frac{\pi}{6} \leqslant \phi \leqslant \frac{\pi}{6}$), где $\phi < 0$ если колеса повернуты вправо;
- dt – время через которое происходят замеры в мс ($-10^6 < dt < 10^6$);
- N — количество произведенных замеров ($0 < N < 10^6$).

Далее идут N строк состоящие из 2 вещественных чисел через пробел: $v,\ \omega$:

- v — линейная скорость перемещения робота в м/с ($0 < v < 10^3$);
- ω — угловая скорость в рад/с ($-1 \leqslant \phi \leqslant 1$).

Формат выходных данных

Два вещественных числа с точностью до сотых через пробел — координаты X, Y центра задней оси после описанного движения.

Примеры

Примеры тестовых полигонов представлены по данной ссылке http://bit.ly/38k4ydK.

Решение

Рассмотрим кинематическую модель управляемой тележки. За основу возьмем модель Аккермана (или принцип рулевого управления Аккермана). Особенность модели Аккермана — центр вращения модели находится на задней оси.

По условиям задачи расстояние между передней и задней осями (колесная база) постоянное и составляет 2,7 метра.

Кинематическая схема представлена на рис.

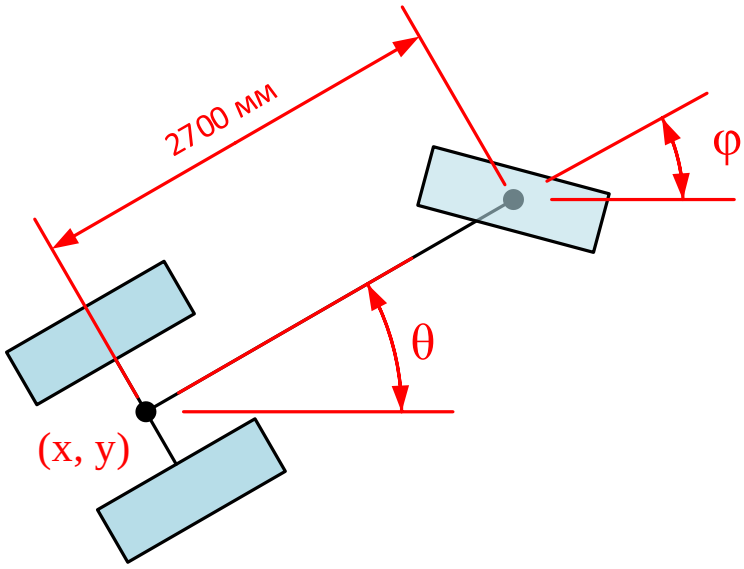


Рис. II.1.18: Кинематическая схема модели Аккермана

Задача прямой кинематики — зная параметры управления (скорости и время) вычислить положения робота в пространстве. В нашем случае — это вычисление текущих координат X; Y и угла поворота относительно начальных координат.

Мобильные роботы с рулевым управлением являются неголономной механической системой.

Голономная механическая система — *число управляемых степеней свободы равно общему количеству степеней свободы.*

Неголономная механическая система — *механическая система, на которую помимо геометрических связей накладываются и кинематические. Математически неголономные связи выражаются неинтегрируемыми уравнениями.*

Уравнение прямой кинематики для управляемой тележки:

$$\dot{x} = v \cdot \cos(\phi)$$
$$\dot{y} = v \cdot \sin(\phi)$$
$$\dot{\theta} = \frac{v}{l} \cdot \tan(\phi)$$

где v — линейная скорость, l — колесная база

По условиям задачи нам известны начальные координаты робота $(x,\ y)$, угол поворота тележки θ и угол поворота рулевого колеса ϕ.

Далее обрабатываем набор данных с линейной (v) и угловой (w) скоростями, которые применяются с интервалом dt для вычисления конечных координат тележки.

При расчетах стоит учитывать то, что угол поворота переднего колеса может быть только в пределах от -30° до 30°

Прямолинейное движение тележки зависит от поворота переднего колеса. Поэтому сначала вычисляем угол поворота переднего колеса. Рассмотрим пример:

Начальные координаты:
x_0 = -18.213 м, y_0 = 52.244 м, θ_0 = -0.795 рад, ϕ_0 = -0.288 рад, dt = 0.063 с

Первая пара данных: v_1 = 36.000 м/с, w_1 = 0.307 рад/с

1. Вычисляем угол поворота переднего колеса:

$$\phi = \phi_0 + w_1 \cdot dt = -0.288 + 0.307 \cdot 0.063 = -0.269 \text{ рад}$$

2. Вычисляем угол поворота задней оси:

$$\theta = \theta_0 + \frac{v_1}{l} \cdot \tan(\phi) \cdot dt = -0.795 + \frac{36}{2.7} \cdot \tan(-0.269) \cdot 0.063 = -0.282 \text{ рад}$$

3. Вычисляем координаты $(x,\ y)$:

$$x = x_0 + v_1 \cdot \cos(\phi) \cdot dt = -18.213 + 36 \cdot \cos(-0.269) \cdot 0.063 = -16.0266 \text{ м}$$

$$y = y_0 + v_1 \cdot \sin(\phi) \cdot dt = 52.244 + 36 \cdot \sin(-0.269) \cdot 0.063 = 51.6412 \text{ м}$$

и далее по такой же схеме продолжаем пересчитывать координаты с учетом всех данных в наборе. Для повышения точности расчетов углов можно брать среднее значение текущего угла и предыдущего.

В качестве ответа выводим конечные координаты робота $(x,\ y)$ с точностью до 0.001.

Пример программы-решения

Ниже представлено решение на языке Python 3

```
import math

x, y, theta, phi, dt, N = map(float, input().split())
dt /= 1000                # перевод в секунды
L = 2.7                   # колесная база, в метрах
```

```
6
7   # в модели Аккермана макс угол поворота колес 30 град в каждую сторону
8   phi_right_max = math.pi/6
9   phi_left_max = -phi_right_max
10
11  iterations = 10
12  subtime = dt / iterations
13
14  for i in range(int(N)):
15      v, w = map(float, input().split())
16
17      for t in range(0, iterations):
18          phi_prev = phi
19          phi += (w * subtime)
20
21          s = v * subtime
22          phi = phi_right_max if phi > phi_right_max else phi
23          phi = phi_left_max if phi < phi_left_max else phi
24
25          if v != 0:
26              theta_delta = s / L * ((math.tan(phi) + math.tan(phi_prev)) / 2)
27              theta_prev = theta
28              theta += theta_delta
29
30              dx = (math.cos(theta) + math.cos(theta_prev)) / 2 * s
31              dy = (math.sin(theta) + math.sin(theta_prev)) / 2 * s
32              x += dx
33              y += dy
34
35  print(round(x, 3), round(y, 3))
```

Задача II.1.1.6. Определение периметра фигур (16 баллов)

Имеется цветное изображение, на котором расположены несколько различных фигур. Данные фигуры могут иметь различную форму и цвет, а также соприкасаться друг с другом. Необходимо определить наибольшую длину контура фигуры. Он считается по количеству пикселей смежных с фигурой. Фон всегда белый и не является фигурой, среди которых необходимо найти наибольшую по длине контура.

Например, на рис. II.1.19 представлено две фигуры: красная и синяя, которые имеют следующую длину контура:

- Длина контура красной фигуры: 10;
- Длина периметра синей фигуры: 34.

Формат входных данных

Входной файл содержит 600 строк, на каждой строке находится 800 шестнадцатеричных чисел — изображение размером 800×600 в виде шестнадцатеричных чисел слева направо сверху вниз. Данные числа имеют следующий вид: $RRGGBB$, где RR — 16тиричное число R составляющей данного элемента матрицы, GG и BB — это 16ти-ричные числа G и B составляющих соответственно.

Все числа являются целыми.

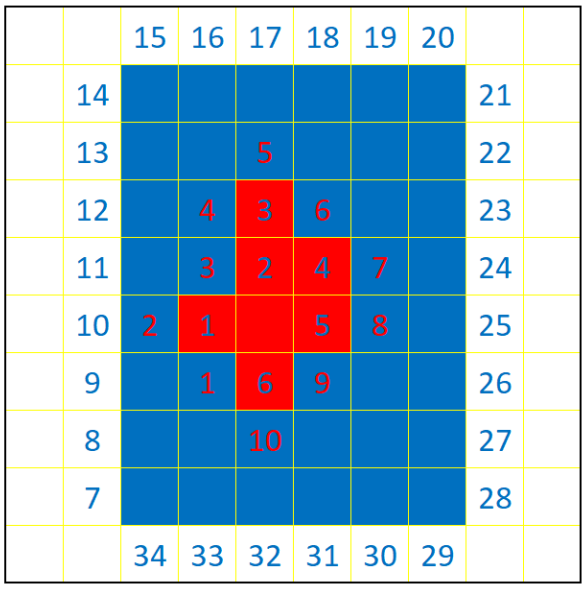


Рис. II.1.19: Пример изображения

Формат выходных данных

Одно целое число — наибольшую длину контура фигуры.

Примеры

Примеры тестовых полигонов представлены по данной ссылке http://bit.ly/34wilwD.

Комментарии

При решении данной задачи запрещено использовать готовые библиотеки определения контуров объектов, например библиотеку OpenCV. Такие решения будут оцениваться в 0 баллов.

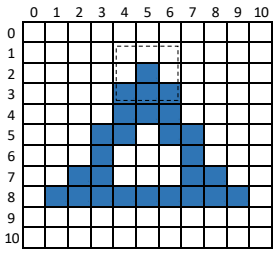
Решение

Начинаем сканирование изображения попиксельно, слева направо, сверху вниз (порядок на самом деле не особо важен, главное — учитывать уже проверенные пиксели, например, добавлять в список или массив их координаты (x, y)). Ищем пиксели любого цвета, кроме белого (по условиям задачи пиксели белого цвета ($FFFFFF$) являются фоном и не могут принадлежать фигурам).

Как только находим пиксель не белого цвета, начинаем поиск соседних пикселей такого же цвета через алгоритм BFS (поиск в ширину). Проверяем только смежные пиксели. Пиксели, находящиеся по диагонали от текущего могут принадлежать другой фигуре, которая может быть такого же цвета!

Подробнее про алгоритм BFS, а также другие алгоритмы поиска, можно посмотреть в презентации “Планирование и построение маршрута” на сайте Университета Иннополис по ссылке: .

Пример работы алгоритма BFS для соседних пикселей показан на рис. II.1.20.



Итерация	Текущий пиксель (x, y)	Очередь для проверки (после добавления новых пикселей)	Список координат проверенных пикселей
1	5, 2	(5,3)	
2	5, 3	(4,3), (6,3), (5,4)	(5,2), (5,3)
3	4, 3	(6,3), (5,4), (4,4)	(5,2), (5,3), (4,3)
4	6, 3	(5,4), (4,4), (6,4)	(5,2), (4,3), (5,3), (6,3)
5	5, 4	(4,4), (6,4)	(5,2), (4,3), (5,3), (6,3), (5,4)
6	4, 4	(6,4), (4,5), (6,5)	(5,2), (4,3), (5,3), (6,3), (5,4), (4,4)
...		и так далее, пока очередь не будет пустая	

Рис. II.1.20: Пример сканирования пикселей фигуры по выбранному цвету

Далее, имея набор пикселей, принадлежащих одной фигуре (одного цвета), нам надо найти длину контура. Контур — пиксели, отличающиеся от цвета фигуры и прилегающие к внешним границам фигуры или к внутренним (если таковые имеются). Учитываются только смежные пиксели! Не забываем учитывать пиксели контура, которые уже проверили от соседних пикселей.

Пример пикселей, составляющих контур фигуры, показан на рис. II.1.21.

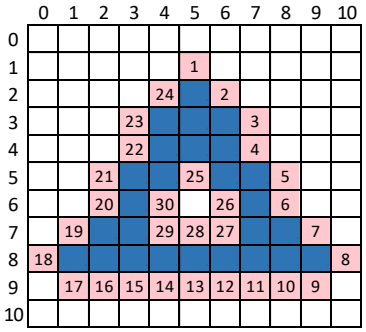


Рис. II.1.21: Длина контура указанной фигуры — 30 пикселей

Подобным образом вычисляя контуры фигур на изображении, выбираем максимальный размер из уже проверенных фигур.

В качестве ответа выводим сумму пикселей фигуры с максимальным количеством.

Пример программы-решения

Ниже представлено решение на языке Python 3

```
def get_max_contour(pic):
    def get_edge_pixels(y, x):
        len_looked_start = len(looked)
        looked1 = [(y + dy, x + dx) for dy, dx in neighs if pic[y+dy][x+dx] !=
        ↪  figure_clr and (y+dy, x+dx) not in looked]
        for i in looked1:
            looked.append(i)

        return len(looked) - len_looked_start


```

```
    visited = [[pic[r][c] == 'ffffff' for c in cols_range] for r in rows_range]
    max_contour = 0

    for row in rows_range:
        for col in cols_range:
            if not visited[row][col]:
                figure_clr = pic[row][col]
                contour = 0
                queue = [(row, col)]
                looked = []
                while queue:
                    r, c = queue.pop(0)

                    if not visited[r][c]:
                        visited[r][c] = True
                        contour += get_edge_pixels(r, c)

                        for dr, dc in neighs:
                            rr = r + dr
                            cc = c + dc
                            if rr in rows_range and cc in cols_range and \
                               pic[rr][cc] == figure_clr and not visited[rr][cc]:
                                queue.append((rr, cc))

                max_contour = max(contour, max_contour)

    return max_contour


################ PROGRAM START ###################

cols_range = range(800)
rows_range = range(600)
PIC = []

for _ in rows_range:
    PIC.append(input().split())

neighs = [(-1, 0), (0, 1), (1, 0), (0, -1)]

result = get_max_contour(PIC)
print(result)
*
```

Задача II.1.1.7. Распознование arTag маркера (16 баллов)

Робот, собранный по дифференциальной схеме, оснащен камерой. Камера позволяет получить изображение arTag метки и ее окружения. Изображение с камеры приходит в управляющую программу в виде набора 160×120 точек, где каждая точка закодирована в RGB-формате.

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

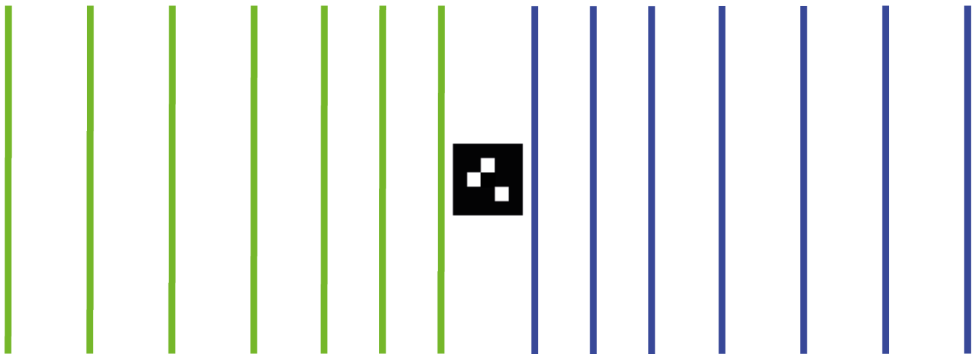


Рис. II.1.22: Общий вид маркера с окружением

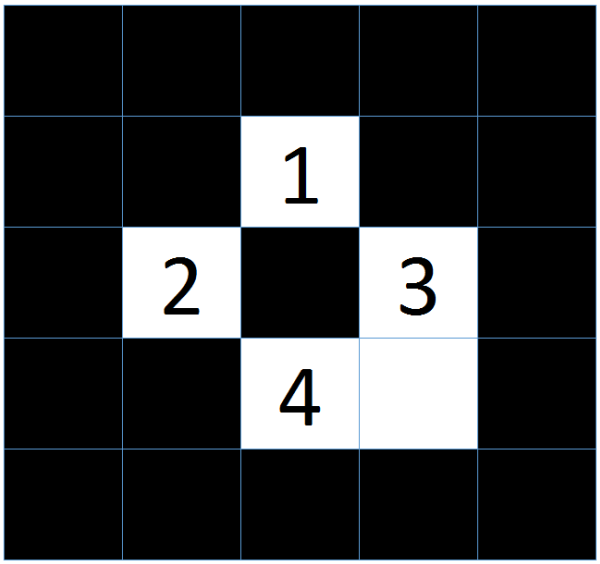


Рис. II.1.23: Нумерация битов в маркера

кере числа нечетное, то он белый. Оставшиеся 4 элемента маркера кодируют число по следующему правилу: если элемент черный, то он обозначает 0, если белый, то 1 при этом самый первый элемент — старший бит закодированного числа. Элементы пронумерованы сверху вниз, слева направо (см. рис. II.1.23).

Поскольку изображение было получено в процессе движения, то изображение ArTag маркера, получаемые с камеры, получаются в виде неправильного выпуклого четырехугольника, а также может оказаться обрезанным. Также известно, что вокруг маркера имеются вертикальные линии, расстояние между которыми уменьшается, приближаясь к маркеру. Линии, расположенные с правой стороны относительно маркера — синие.

Формат входных данных

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

Все числа являются целыми.

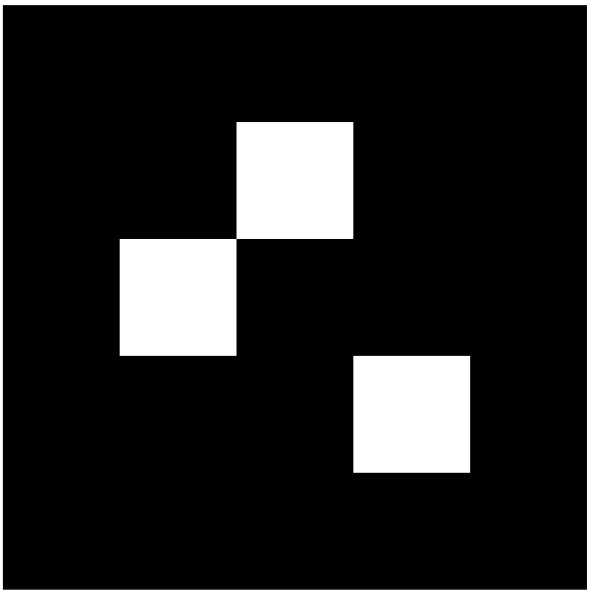


Рис. II.1.24: Маркер с закодированным значением — $0011_2 = 3_{10}$

Формат выходных данных

Одна строка, содержащая значение маркера в десятичной системе. В случае если значение маркера нельзя определить необходимо вывести следующее:

- L — если роботу необходимо повернуть влево, чтобы увидеть маркер полностью;
- R — если роботу необходимо право.

Примеры

Примеры тестовых полигонов представлены по данной ссылке `http://bit.ly/2WzkeUU`.

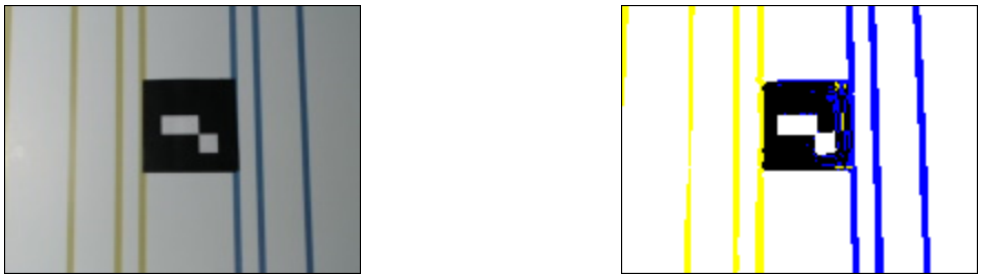
Решение

Подробное описание по обработке изображений и распознаванию меток ArTag можно посмотреть в презентации “Распознавание ARTag” в разделе “Алгоритмические основы технического зрения” на сайте Университета Иннополис по ссылке: http://bit.ly/IRS_info.

Для обработки изображений в компьютерном зрении часто используют цветовую модель HSV (Hue — тон, Saturation — насыщенность и Value — значение) по причине более удобной программной обработки, по сравнению с моделью RGB. После перевода всего изображения в формат HSV, переведем значение каждого пикселя в “номер цвета”, например, 1 — черный цвет, 2 — синий цвет, 3 — зеленый цвет, 4 — желтый цвет, 5 — красный цвет, 6 — белый цвет. При работе с моделью HSV мы можем очень гибко и точно настраивать цветовые диапазоны для каждого цвета.

Исходное изображение в формате RGB и HSV (с нумерацией цветов) показаны на II.1.25.

На примере изображения RGB заметно, что изображение с пониженной ярко-



RGB HSV

Рис. II.1.25: Исходные изображения в форматах RGB и HSV-color

стью и контрастностью. Это может быть вызвано многими факторами — характеристиками камеры, неправильными настройками камеры, пониженной освещенностью окружения в момент съемки и т. д. Из-за этого у нас получается некорректное распознавание номеров цветов (черный и синий вперемешку). Поэтому, для дальнейшей обработки изображения мы программно увеличим контрастность и немного уменьшим яркость. Результат обработки показан на рис. II.1.26.

RGB HSV

Рис. II.1.26: Обработанные изображения в форматах RGB и HSV-color

Теперь на изображении четко заметны правильные границы всех цветов. По условиям задачи нам нужно декодировать ArTag-метку, если это возможно. Если же метка в кадре отсутствует или она не полностью, то надо написать, в какую сторону должен повернуться робот, чтобы повторить съемку метки.

В нашем случае изображение в формате HSV-color представляет двумерный массив из номеров цветов. Пример фрагмента массива показан на рис.II.1.27.

Сейчас пробуем декодировать метку ArTag. Границы метки определяем по цветности "1". По условиям задачи черный цвет метки = "0 белый = "1". В нашем примере получается двоичное число 0010_2 и ответом будет "2"

Варианты ситуаций, когда метка в кадре не полностью или вообще отсутствует:

- Не получается раскодировать метку (не проходит проверка бита четности — parity check)
- На изображении нет синих вертикальных полос ⇒ правая часть метки не в кадре
- На изображении нет желтых/зеленых вертикальных полос ⇒ левая часть метки не в кадре
- На изображении отсутствуют пиксели черного цвета ⇒ метки вообще нет в кадре

В случае, когда метки нет к кадре или ее невозможно декодировать, определяем в какую сторону должен повернуть робот, чтобы “увидеть” метку. Как вариант,

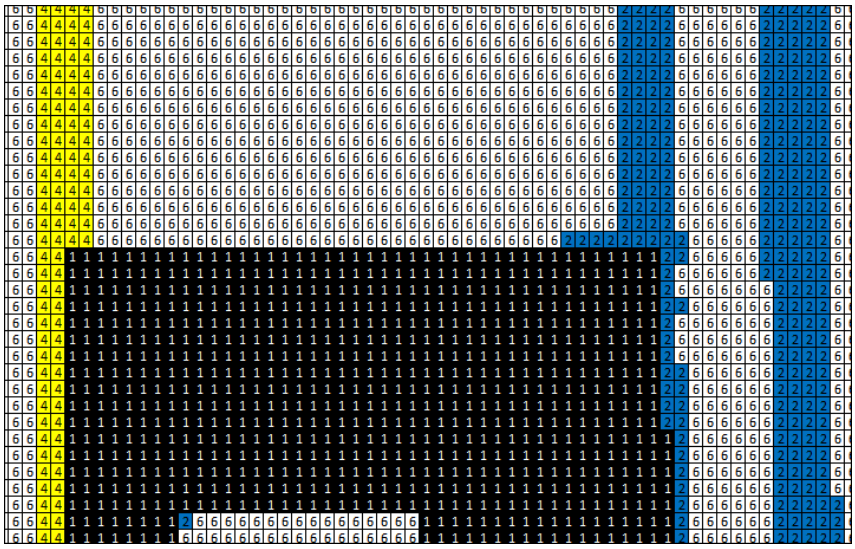
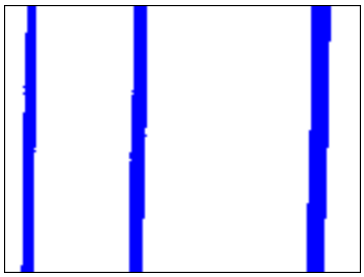
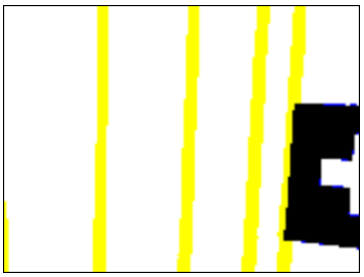


Рис. II.1.27: Фрагмент массива данных с номерами цветов

сторону движение можно определить по количеству пикселей цветных полосок. Если больше пикселей синего цвета, то надо повернуть налево, в этом случае ответом будет "L", если же больше пикселей желтого/зеленого цвета, то робот должен повернуть направо, в этом случае ответом будет "R".



Нужен поворот налево



Нужен поворот направо

Рис. II.1.28: Примеры отсутствия или неполного размещения метки ArTag в кадре

Пример программы-решения

Ниже представлено решение на языке Python 3

```
class Artag:
    def __init__(self, sections = 5, image_bw = [], radius = 0, border = 0):
        self.sections = sections
        self.total_bits = (sections - 2) ** 2 - 4
        self.img_bw = image_bw
        self.radius = radius
        self.img_height = len(image_bw)
        self.img_width = len(image_bw[0])
        self.neighs = [(-1, 0), (0, 1), (1, 0), (0, -1)]
        self.border = border

        self.control_bits = [2 ** i - 1 for i in range(self.total_bits + 1)][:sections
        ↪  - 2]
        if sections == 5:
            self.control_bits = [2]
```

```
15
16          self.info_bits = { 5: {'N':(0, 1, 2, 3)},
17                             6: {'N':(2, 4), 'X':(5, 6, 8), 'Y':(9, 10, 11)},
18                             8: {}
19                           }
20          if sections == 8:
21              controls_bits = set(self.control_bits)
22              info_bits = list(set(range(self.total_bits)).difference(controls_bits))
23              for n in range(len(info_bits))[::2]:
24                  self.info_bits[sections][f'N{int(n/2+1)}'] = (info_bits[n],
                    ↪  info_bits[n+1])
25
26
27      def get_contour(self):
28          def pixel_on_edge(y, x):
29              pixels_on_edge = sum([1 for dy, dx in self.neighs if
                ↪  self.img_bw[y+dy][x+dx] != figure_clr])
30              return pixels_on_edge > 0
31
32          range_rows = range(self.img_height)
33          range_cols = range(self.img_width)
34
35          visited = [[True if self.img_bw[r][c] == 0 else False for c in range_cols] for
            ↪  r in range_rows]
36
37          for r in range(self.border):
38              visited[r] = [True] * self.img_width
39              visited[self.img_height-r-1] = [True] * self.img_width
40
41          for r in range(self.img_height):
42              visited[r][:self.border] = [True] * self.border
43              visited[r][self.img_width - self.border:] = [True] * self.border
44
45          figure_clr = 1
46          contour = []
47          for row in range_rows:
48              for col in range_cols:
49                  if not visited[row][col]:
50                      looked = []
51                      queue = [(row, col)]
52                      while queue:
53                          r, c = queue.pop(0)  #row, col
54
55                          if not visited[r][c]:
56                              visited[r][c] = True
57                              if pixel_on_edge(r, c) and (r, c) not in contour:
58                                  contour.append((r, c))
59
60                              for dr, dc in self.neighs:
61                                  rr = r + dr
62                                  cc = c + dc
63                                  if rr in range_rows and cc in range_cols and \
64                                     self.img_bw[rr][cc] == figure_clr and not
                                       ↪  visited[rr][cc] and (rr,cc) in contour:
65                                          queue.append((rr, cc))
66          return contour
67
68      def lines_intersection(self, line1, line2):
69          y1, x1 = line1[0]
70          y2, x2 = line1[1]
```

```
71          y3, x3 = line2[0]
72          y4, x4 = line2[1]
73
74          x =
            ↪  -(((x1*y2-x2*y1)*(x4-x3)-(x3*y4-x4*y3)*(x2-x1))/((y1-y2)*(x4-x3)-(y3-y4)*(x2-x1)))
75          y = ((y3-y4)*(-x)-(x3*y4-x4*y3))/(x4-x3)
76
77          return int(y), int(x)
78
79      def line_split(self, x1y1, x2y2, sections = 2):
80          out = [x1y1]
81          dY = (x2y2[0] - x1y1[0]) / sections
82          dX = (x2y2[1] - x1y1[1]) / sections
83
84          y1, x1 = x1y1
85          out += [[round(y1 + dY * i), round(x1 + dX * i)] for i in range(1, sections)]
86          out.append(x2y2)
87
88          return out
89
90      def get_point(self, y, x):
91          if y not in range(len(self.img_bw)) or x not in range(len(self.img_bw[0])):
92              return False
93
94          if self.radius == 0:
95              out = self.img_bw[y][x]
96          else:
97              out = [[self.img_bw[i][j] for j in range(x-self.radius, x+self.radius+1)]
                    ↪  \
98                      for i in range(y-self.radius, y+self.radius + 1)]
99
100         return out
101
102
103     def get_pixels_by_xy(self, centers):
104         out = []
105         for y, x in centers:
106             pixels = sum(self.get_point(y, x), [])
107             if pixels == False:
108                 print('Pixels out of range')
109                 break
110             pixels = list(pixels) if type(pixels) == int else pixels
111
112             out.append(round(sum(pixels) / len(pixels)))
113
114         return out
115
116     def array_x1_to_x2(self, arr, size):
117         rows, cols = size
118         return [arr[r*rows: r*rows+cols] for r in range(rows)]
119
120     def array_rotate(self, arr):
121         arr_h, arr_w = len(arr), len(arr[0])
122         xx = 0
123         out = []
124         for y in range(arr_h):
125             out.append([])
126             yy = arr_h - 1
127             for x in range(arr_w):
128                 out[y].append(arr[yy][xx])
```

```
129                 yy -= 1
130             xx += 1
131
132         return out
133
134     def get_numbers(self, bits):
135         out = {}
136         out_nums = []
137         numbers = self.info_bits[self.sections]
138         for key in numbers:
139             num = [bits[b] for b in numbers[key]]
140             out[key] = int(''.join(map(str, num)), 2)
141             out_nums.append(out[key])
142
143         return out
144
145     def decode(self):
146         contour_pixels = self.get_contour()
147
148         yy = [y for y, x in contour_pixels]
149         xx = [x for y, x in contour_pixels]
150         min_y, max_y = min(yy), max(yy)
151         min_x, max_x = min(xx), max(xx)
152         #min_y = min(contour_pixels, key = lambda yx: yx[0])[0]
153
154         A = min([(y,x) for y,x in contour_pixels if y == min_y], key = lambda yx:
            ↪ yx[1])   # min_x -> min_x
155         B = min([(y,x) for y,x in contour_pixels if x == max_x], key = lambda yx:
            ↪ yx[0])   # max_x -> min_y
156         C = max([(y,x) for y,x in contour_pixels if y == max_y], key = lambda yx:
            ↪ yx[1])   # max_y -> max_x
157         D = max([(y,x) for y,x in contour_pixels if x == min_x], key = lambda yx:
            ↪ yx[0])   # min_x -> max_y
158         abcd = [A, B, C, D]
159
160         center = self.lines_intersection([A, C], [B, D])
161         setka = self.sections * 2
162
163         edge_points = [self.line_split(abcd[i], abcd[(i+1)%4], setka) for i in
            ↪ range(4)]
164
165         CD = edge_points[2][::-1]
166         DA = edge_points[3][::-1]
167         lines_x = [self.line_split(DA[i], edge_points[1][i], setka) for i in
            ↪ range(setka)]
168         lines_y = [self.line_split(edge_points[0][i], CD[i], setka) for i in
            ↪ range(setka)]
169
170         centers = [[lines_x[y][x] for x in range(3, setka-2, 2)] for y in range(3,
            ↪ setka-2, 2)]
171         centers = sum(centers, [])
172
173         bits = self.get_pixels_by_xy(centers)
174
175         sections_inside = self.sections - 2
176
177         bits_inside = self.array_x1_to_x2(bits, [sections_inside, sections_inside])
178
179         for i in range(4):
180             if bits_inside[-1][-1] == 0: break
```

```
181             bits_inside = self.array_rotate(bits_inside)
182
183         bits_inside[0]  = bits_inside[0][1:-1]
184         bits_inside[-1] = bits_inside[-1][1:-1]
185
186         bits = sum(bits_inside, [])
187
188         out = []
189         if self.sections == 5:
190             parity = bits.pop(2)                                    # убираем бит
                ↪  четности
191             if bits.count(1) % 2 != parity:
192                 print('Parity not good !!!')
193                 return False
194
195         elif self.sections in (6, 8):
196             hem = Hemming(self.sections)
197             bits = hem.decode(bits)                             # проверка по Хэммингу
                ↪  #и удаление контрольных битов
198
199         out = self.get_numbers(bits)
200         return out
201
202 class Images:
203     def __init__(self, width = 160, height = 120):
204         self.rgb = []
205         self.bw = []
206         self.grey = []
207         self.rgb24 = []
208         self.hex = []
209         self.hsv = []
210         self.w = width
211         self.h = height
212
213     def hsv_rgb(self, hsv):
214         h, s, v = hsv
215         s /= 100
216         v /= 100
217         c = v * s
218         x = c * (1 - abs((h / 60) % 2 - 1))
219         m = v - c
220
221         rgb = []
222         if h < 60:
223             rgb = [c, x, 0]
224         elif h < 120:
225             rgb = [x, c, 0]
226         elif h < 180:
227             rgb = [0, c, x]
228         elif h < 240:
229             rgb = [0, x, c]
230         elif h < 300:
231             rgb = [x, 0, c]
232         else:
233             rgb = [c, 0, x]
234
235         rgb = [round((ch + m) * 255) for ch in rgb]
236
237         return rgb
238
```

```
239     def brightness(self, img_rgb, value = 1.5):
240         img_out = []
241         for row in range(len(img_rgb)):
242             img_out.append([])
243             for col in range(len(img_rgb[0])):
244                 img_out[-1].append([int(min(255, max(0, ch * value))) for ch in
                    ↪  img_rgb[row][col]])
245         return img_out
246 
247     def contrast(self, img_rgb, value = 100):
248         level = (100 + value) / 255          # Вычисляем общее значение
249         level *= level
250 
251         img_out = []
252         for h in range(len(img_rgb)):
253             img_out.append([])
254             for w in range(len(img_rgb[0])):
255                 rgb = [int(((ch/255-0.5) * level + 0.5) * 255) for ch in
                    ↪  img_rgb[h][w]]
256 
257                 for ch in range(len(rgb)):
258                     rgb[ch] = 0 if rgb[ch] < 0 else rgb[ch]
259                     rgb[ch] = 255 if rgb[ch] > 255 else rgb[ch]
260 
261                 img_out[-1].append(rgb)
262 
263         return img_out
264 
265     def get_color_from_hsv(self, pixel_hsv):
266         # 1 - black, 2 - blue, 3 - green, 4 - yellow, 5 - red, 6 - white
267         h, s, v = pixel_hsv
268 
269         color = -1
270         colors = [ (90, 4), (300, 2), (360, 6)]          # (h, color)
271 
272         if s > 20:
273             for h_limit, clr in colors:
274                 if h < h_limit:
275                     color = clr
276                     break
277         else:
278             color = 1 if v < 10 else 6
279 
280         return color
281 
282 
283     def rgb_hsv(self, pixel_rgb):
284         pixel_rgb = [pix/255 for pix in pixel_rgb]
285         r, g, b = pixel_rgb
286         max_rgb = max(pixel_rgb)
287         min_rgb = min(pixel_rgb)
288         delta = max_rgb - min_rgb
289 
290         h, s = 0, 0
291 
292         if delta > 0:
293             if r == max_rgb and g >= b:
294                 h = 60 * ((g - b) / delta)
295             elif r == max_rgb and g < b:
296                 h = 60 * ((g - b) / delta) + 360
```

```
297             elif g == max_rgb:
298                 h = 60 * ((b - r) / delta) + 120
299             elif b == max_rgb:
300                 h = 60 * ((r - g) / delta) + 240
301 
302         if max_rgb > 0:
303             s = (delta / max_rgb) * 100
304 
305         v = max_rgb * 100
306 
307         hsv = [h, s, v]
308         #print('hsv[%d %5.3f %d], rgb[%5.3f %5.3f %5.3f]' % (h,s,v, r,g,b))
309         return self.get_color_from_hsv(hsv)
310 
311     def hex_rgb(self, pixel):
312         rgb = [pixel[p:p+2] for p in range(len(pixel))[::2]]
313         return list(map(lambda x: int(x, 16), rgb))
314 
315     def rgb24_rgb(self, clr24):
316         return [(clr24 & 0xFF0000) >> 16, (clr24 & 0x00FF00) >> 8, clr24 & 0x0000FF]
317 
318     def rgb_rgb24(self, pixels):
319         #pixels.reverse()
320         return sum([pixels[p] * 256 ** p for p in range(2, -1, -1)])
321 
322     def update(self, source = 'hex'):
323         # schemes = ['r_g_b', 'bw', 'grey', 'rgb', 'hex']
324         if source == 'hex':
325             self.rgb   = [list(map(self.hex_rgb, self.hex[row])) for row in
                ↪  range(img.h)]
326             self.rgb24 = [list(map(lambda x: int(x, 16), self.hex[row])) for row in
                ↪  range(img.h)]
327             self.hsv   = [list(map(self.rgb_hsv, self.rgb[row])) for row in
                ↪  range(img.h)]
328 
329         elif source == 'rgb24':
330             self.rgb = [list(map(self.rgb_rgb, self.rgb24[row])) for row in
                ↪  range(img.h)]
331             self.hex = [list(map(hex, self.rgb24[r])) for row in range(img.h)]
332             self.hsv = [list(map(self.rgb_hsv, self.rgb24[row])) for row in
                ↪  range(img.h)]
333 
334         elif source == 'rgb':
335             self.rgb24 = [list(map(self.rgb_rgb24, self.rgb[row])) for row in
                ↪  range(img.h)]
336             self.hex   = [list(map(hex, self.rgb24[row])) for row in range(img.h)]
337             self.hsv   = [list(map(self.rgb_hsv, self.rgb[row])) for row in
                ↪  range(img.h)]
338 
339 
340 def hsv_rgb(img_hsv, colors = [1, 2, 3, 4, 5, 6]):
341     out = []
342     colors_rgb = {1: [0,0,0], 2: [0,0,255], 3: [0,255,0], 4: [255,255,0], 5:
        ↪  [255,0,0], 6: [255,255,255]}
343 
344     for row in range(len(img_hsv)):
345         out.append([])
346         for col in range(len(img_hsv[0])):
347             out[-1].append(colors_rgb[6] if img_hsv[row][col] not in colors else
                ↪  colors_rgb[img_hsv[row][col]])
```

```
348
349     return out
350
351
352 ################ PROGRAM START ###################
353
354 img = Images(width=160, height=120)
355
356 #with open('01') as f:
357 #    data = f.readlines()
358 #img.hex = [data[r].strip().split() for r in range(img.h)]
359 img.hex = [input().split() for _ in range(img.h)]
360
361 img.update(source = 'hex')
362 img.rgb = img.brightness(img.rgb, 0.9)
363 img.rgb = img.contrast(img.rgb, 255)
364 img.update(source = 'rgb')
365 img_bw = []
366 for row in img.hsv:
367     img_bw.append([1 if pix == 1 else 0 for pix in row])
368
369 artag = Artag(sections = 5, image_bw = img_bw)
370 result = artag.decode()
371
372 if result == -1:
373     result = 'R'
374     blue_pixels = sum(img.hsv, []).count(2)
375     yellow_pixels = sum(img.hsv, []).count(4)
376     if blue_pixels > yellow_pixels:
377         result = 'L'
378
379 print(result)
```

Задача II.1.1.8. Перемещение в смежный сектор (20 баллов)

Два автоматизированных погрузчика были активированы в разных помещениях логистического центра. У каждого робота в постоянной памяти есть карта всего логистического центра и информация о местоположении обоих автоматизированных погрузчиков. Их работа настроена таким образом, что время можно измерять в “фазах перемещения”. За одну “фазу перемещения” погрузчик переходит прямо из одного помещения в другое, либо выполняет поворот на 90° внутри текущего помещения.

Логистический центр представляет собой полигон (рис. II.1.29) размером 12×12 помещений. Помещения представляют из себя квадратные секции. Некоторые помещения могут быть доступны только из определенных соседних секций, так и недоступны вовсе. Отсчет координат начинается с левого верхнего угла, где ось Y направлена вниз, ось X направлена вправо.

Первый погрузчик постоянно перемещается по “правилу левой руки”, то есть движется вдоль левой стенки, когда спереди есть свободное помещение. Если слева стенки нет, то погрузчик должен повернуть налево на 90 градусов и проехать одно помещение вперед. Если стенка есть слева и спереди, то погрузчик поворачивает направо на 90 градусов.

Второй погрузчик может передвигаться в любом направлении. Необходимо настроить его перемещение таким образом, чтобы он как можно быстрее встретился с первым погрузчиком. Под встретился подразумевается то, что в какой-то момент

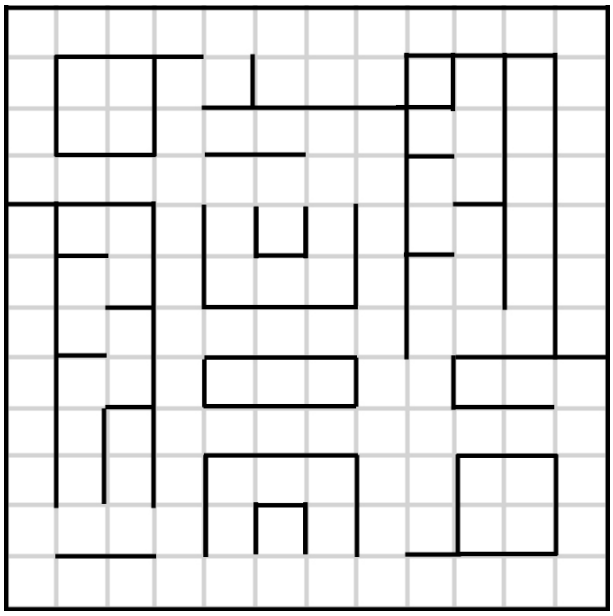


Рис. II.1.29: Карта логистического центра

времени оказался в соседнем достижимом секторе возле первого погрузчика.

Направление положения погрузчика задается одной из четырех букв:

- U — погрузчик направлен вверх (в сторону отрицательного направления оси Y);
- L — погрузчик направлен влево (в сторону отрицательного направления оси X);
- D — погрузчик направлен вниз (в сторону положительного направления оси Y);
- R — погрузчик направлен вправо (в сторону положительного направления оси X);

Формат входных данных

Первая строка входных данных содержит 2 целых числа y_1, x_1 и одну заглавную букву dir_1, через пробел, где:

- y_1 — координата первого погрузчика по оси Y ($1 \leqslant y_1 \leqslant 12$);
- x_1 — координата первого погрузчика по оси X ($1 \leqslant x_1 \leqslant 12$);
- dir_1 — направление первого погрузчика ($dir_1 \in U, R, D, L$);

Вторая строка входных данных содержит 2 целых числа y_2, x_2 и одну заглавную букву dir_2, через пробел, где:

- y_2 — координата второго погрузчика по оси Y ($1 \leqslant y_2 \leqslant 12$);
- x_2 — координата второго погрузчика по оси X ($1 \leqslant x_2 \leqslant 12$);
- dir_2 — направление второго погрузчика ($dir_2 \in U, R, D, L$);

Формат выходных данных

Выведите единственное число — минимальное возможное время на достижение вторым погрузчиком смежной клетки с первым (количество “фаз перемещения”).

Примеры

Пример №1

Стандартный ввод
7 6 D 1 8 D
Стандартный вывод
15

Пример №2

Стандартный ввод
7 1 D 1 7 D
Стандартный вывод
31

Комментарии

Гарантируется, что первый погрузчик достижим вторым роботом.

Решение

По условиям задачи мы знаем секторы старта и направление обоих погрузчиков. Первый погрузчик (далее — П1) перемещается в лабиринте по алгоритму «Правило левой руки». Алгоритм показан на рис. II.1.30. Будьте внимательны — начальные координаты указываются исходя из диапазона секторов от 1 до 12 и первой координатой указывается y!

Поле лабиринта известно заранее и неизменно для всех тестов. Варианты представления карты известного лабиринта (матрица смежности, список ребер) и поиск оптимального пути между двух секторов можно посмотреть в презентации "Планирование и построение маршрута"на сайте Университета Иннополис по ссылке: http://bit.ly/IRS_info.

Для решения задачи мы будем после каждого действия робота П1 (движение прямо и поворот — два разных действия) составлять путь проезда робота П2 до смежных секторов с роботом П1. Для этого надо постоянно обновлять координаты координаты и направление робота П1 для построения оптимального маршрута перемещения роботом П2. Количество действий, совершенных роботом П1, считаем с самого начала.

Получаемые пути перемещения робота П2 к роботу П1 переводим в команды-действия ’F’,’R’,’L’ (или просто подсчитываем количество команд). Если количество

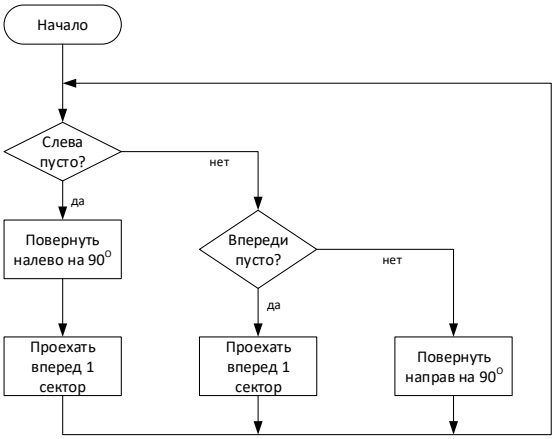


Рис. II.1.30: Алгоритм “Правило левой руки” для перемещения в лабиринте

команд робота П2 совпадет с количеством совершенных действий роботом П1 и при этом роботы окажутся в смежных секторах (между которыми нет стенок!), то задача получается выполненной, а в качестве ответа выводим количество операций.

На рис. II.1.31 состояние роботов на старте.

Рис. II.1.31: Состояние роботов на старте

На рис. II.1.32 показаны промежуточные состояния роботов в момент, когда робот П1 уже совершил 5 действий (на рисунке — схема слева) и когда роботы оказались в смежных секторах с равным количество действий (схема справа).

Следовательно, в данном примере правильный ответ: 8.

Пример программы-решения

Ниже представлено решение на языке Python 3

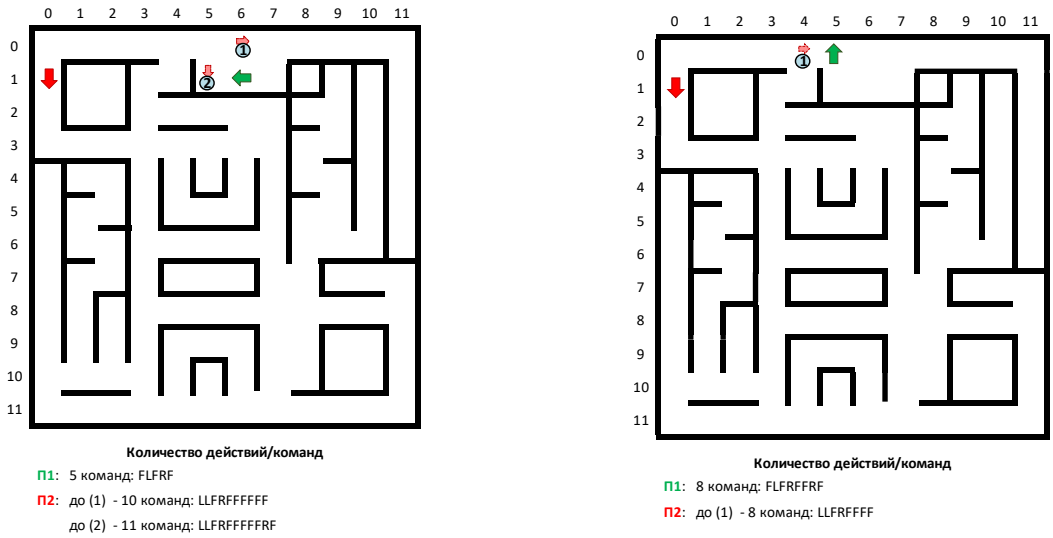


Рис. II.1.32: Состояние роботов через 5 действий П1 (слева) и 8 действий П1 (справа)

```
adj = [[1, 12], [2, 0], [3, 1], [4, 2], [5, 16, 3], [6, 17, 4], [7, 18, 5], [8, 19,
↪  6], [9, 7], [10, 8], [11, 9], [23, 10], [0, 24], [14, 25], [26, 13], [16, 27], [4,
↪  15], [5, 18], [6, 19, 17], [7, 18], [], [33], [34], [11, 35], [12, 36], [13, 26],
↪  [14, 25], [15, 28, 39], [29, 27], [30, 28], [31, 42, 29], [43, 30], [33], [21, 45,
↪  32], [22, 46], [23, 47], [24, 37], [38, 36], [39, 37], [27, 40, 51, 38], [41, 52,
↪  39], [42, 53, 40], [30, 43, 54, 41], [31, 55, 42], [45, 56], [33, 44], [34, 58],
↪  [35, 59], [60], [50], [62, 49], [39, 63], [40, 64], [41], [42, 66], [43, 67], [44,
↪  57], [69, 56], [46, 70], [47, 71], [48, 72], [62, 73], [50, 61], [51, 75], [52,
↪  65], [66, 64], [54, 65], [55, 79], [69, 80], [57, 81, 68], [58, 82], [59, 83],
↪  [60, 84], [61, 74], [86, 73], [63, 76, 87], [77, 75], [78, 76], [79, 77], [67, 91,
↪  78], [68, 81, 92], [69, 82, 80], [70, 81], [71], [72, 96], [86, 97], [74, 85],
↪  [75, 99], [89], [90, 88], [89], [79, 92, 103], [80, 104, 91], [94], [95, 93],
↪  [107, 94], [84, 108], [85, 109], [110], [87, 100, 111], [101, 99], [102, 100],
↪  [103, 101], [91, 104, 115, 102], [92, 105, 116, 103], [106, 104], [107, 105], [95,
↪  119, 106], [96, 120], [97, 121], [98, 122], [99, 123], [113, 124], [114, 112],
↪  [126, 113], [103, 116, 127], [104, 128, 115], [118, 129], [130, 117], [107, 131],
↪  [108, 121, 132], [109, 122, 120], [110, 123, 121], [111, 135, 122], [112, 136],
↪  [137], [114, 138], [115, 128, 139], [116, 127], [117, 130], [118, 129], [119,
↪  143], [120, 133], [134, 132], [135, 133], [123, 136, 134], [124, 137, 135], [125,
↪  138, 136], [126, 139, 137], [127, 140, 138], [141, 139], [142, 140], [143, 141],
↪  [131, 142]]

matrix = [[0] * 144 for _ in range(144)]
for i in range(len(adj)):
    for cell in adj[i]:
        matrix[i][cell] = 1
        matrix[cell][i] = 1

def bfs(start, end):
    queue, visited = [start], []
    len_matrix = range(len(matrix))
    while queue:
      p = queue.pop(0)

      if p not in visited:
        visited.append(p)
        queue += [v for v in len_matrix if matrix[p][v] > 0 and v not in visited]

      if p == end: break
```

```
20
21      path_back = [visited[-1]]
22      for v in reversed(visited):
23          path_back.append(v) if matrix[v][path_back[-1]] else False
24
25      return path_back[::-1]
26
27
28  def path_to_actions(path, current_dir):
29      actions = ''
30      variants = {
31                  MAZE_SIZE:  ['LL','R','','L'],
32                   1: ['R','','L','LL'],
33                  -1: ['L','LL','R',''],
34                  -MAZE_SIZE: ['','L','LL','R']
35                  }
36
37      for i in range(1, len(path)):
38        new_dir = path[i] - path[i-1]
39        actions += variants[new_dir][current_dir]
40        current_dir = variants[new_dir].index('')
41        actions += 'F'
42
43      return actions
44
45
46  def check_robot2():
47
48      robot1_neighs = [robot1_cell + d for d in deltas if robot1_cell + d in total_cells
        ↪  and matrix[robot1_cell][robot1_cell + d]]
49
50      if robot2_cell in robot1_neighs:
51          return True
52
53      for neigh in robot1_neighs:
54          path2 = path_to_actions(bfs(robot2_cell, neigh), robot2_az)
55
56          if len(path2) <= len(path1):
57             return True
58
59      return False
60
61  def get_coords(data):
62      data = data.split()
63      data[2] = 'URDL'.index(data[2])
64      y, x, az = map(int, data)
65      return x-1, y-1, az, (y-1) * MAZE_SIZE + (x-1)
66
67
68  ################ PROGRAM START ###################
69  MAZE_SIZE = 12
70  total_cells = range(MAZE_SIZE ** 2)
71  deltas = [-MAZE_SIZE, 1, MAZE_SIZE, -1]
72
73  new_az1 = lambda a: (robot1_az + a) % 4
74
75  robot1_x, robot1_y, robot1_az, robot1_cell = get_coords(input())
76  robot2_x, robot2_y, robot2_az, robot2_cell = get_coords(input())
77
78  steps = 0
```

```
path1 = ''

while True:
   if check_robot2(): break

   neighs1 = [robot1_cell + d for d in deltas]
   is_left_free  = matrix[robot1_cell][neighs1[(robot1_az - 1) % 4]] if
   ↪  neighs1[(robot1_az - 1) % 4] in total_cells else False
   is_front_free = matrix[robot1_cell][neighs1[robot1_az]] if neighs1[robot1_az] in
   ↪  total_cells else False

   if len(path1) > 0 and path1[-1] == 'L':
       path1 += 'F'
       robot1_cell = neighs1_prev[robot1_az]
   else:
       if is_left_free:
           path1 += 'L'
           robot1_az = new_az1(-1)
       else:
           if is_front_free:
               path1 += 'F'
               robot1_cell = neighs1[robot1_az]
           else:
               path1 += 'R'
               robot1_az = new_az1(1)

   neighs1_prev = neighs1
   steps += 1

print(steps)
```