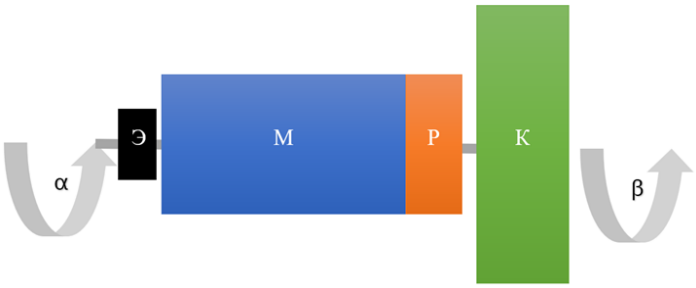
Второй отборочный этап

Младшая возрастная группа

Задача A. Период тиков энкодера

Описание робота:

Привод робота



Э – энкодер. Основная характеристика – количество «тиков» на один оборот (n).

М – мотор. Энкодер вращается на одном валу с мотором и измеряет его угол поворота α.

Р – редуктор. Основная характеристика – передаточное число (i), показывает во сколько раз выходной вал вращается медленнее, чем входной.

К – колесо. Основная характеристика – диаметр (D). Вращается в i раз медленнее, чем мотор. Пока мотор и энкодер делают поворот на α градусов, колесо поворачивается на β градусов.

Задание:

Робот совершает движение с заданной скоростью V. Необходимо вычислить период времени между тиками энкодера (t) на этой скорости, в микросекундах.

Система оценки:

Баллы за каждую подзадачу начисляются только в случае, если все тесты для этой подзадачи и необходимых подзадач успешно пройдены.

Подзадача	Баллы	Ограничения	Необходимые группы тестов
1	20	N=1	-
2	20	n=2	-
3	20	i=1	-
4	20	Основные ограничения	1, 2, 3

Формат входных данных:

Первая строка содержит целое положительное число D ($1 \le D \le 10^3$) — диаметр колеса в миллиметрах.
Вторая строка содержит дробное положительное число i ($10^{-4} \le i \le 10^4$) — редукция, передаточное число между мотором и колесом. Показывает, во сколько раз колесо вращается медленнее, чем мотор.

Третья строка содержит целое положительное четное число n ($2 \le n \le 1440$) — количество «тиков» энкодера на один оборот мотора.

Четвёртая строка содержит целое положительное число N ($1 \le N \le 3000$) — скорость вращения мотора, в оборотах в секунду.

Формат выходных данных:

Выведите целое положительное число - время между тиками энкодера t в микросекундах. При необходимости используется округление вверх.

Решение

Линейная скорость движения робота вычисляется по формуле:

$V_{роб}=w_к*R_к=(w_м*D)/2$

Здесь $w_к$ - угловая скорость колеса, которая вычисляется по формуле:

$w_к=w_м/i$

С ее учетом формула для вычисления линейной скорости приобретает вид:

$V_{роб}=(w_м*D)/2=(D*w_м)/(2*i)$

Выразим из нее угловую скорость мотора:

$w_м=(2*V_{роб}*i)/D$

С другой стороны, угловая скорость переводится из частоты вращения следующим образом:

$w=2*\pi*n/60$ - при n указывающей на количество оборотов в минуту,

$w=2*\pi*n$ - при n указывающей на количество оборотов в секунду.

В нашей задаче с помощью n обозначается количество "тиков" энкодера на один оборот мотора, а скорость в оборотах в секунду с помощью N. Подставим ее в формулу и приравняем с предыдущей, выражающей угловую скорость:

$2*\pi*N=(2*V_{роб}*i)/D$

$\pi*N=(V_{роб}*i)/D$

Теперь рассмотрим подробнее скорость вращения N, выраженную в оборотах в секунду. Это означает, что вал мотора делает N оборотов за секунду, то есть $360*N$ градусов в секунду. Если выразить угол с помощью α, а время с помощью t, то получим формулу:

$N = \alpha / (360*t)$

Ну и вычислим угол α, на который поворачивается вал энкодера между подряд идущими "тиками":

$\alpha = 360 / n$

Объединяем формулы:

$N = 360 / (360 * t * n) = 1/(t*n)$

$\pi/(t*n) = V_{роб}*i/D$ и выражаем из формулы искомое t (в микросекундах):

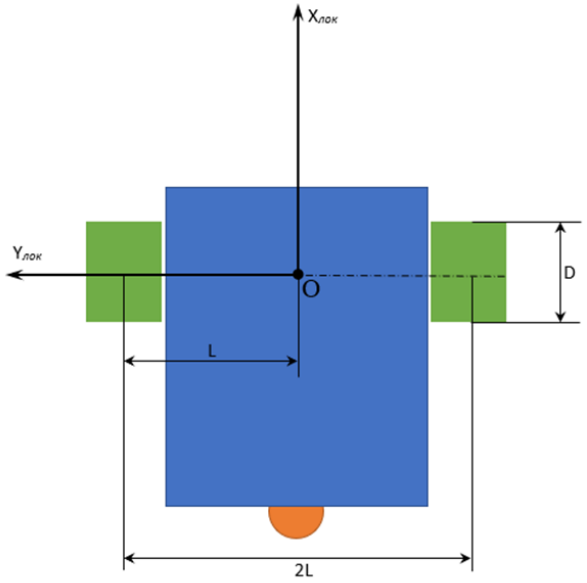
$t=1\ 000\ 000*(\pi*D)/(V_{роб}*i*n)$.

Остается лишь реализовать программное вычисление искомого значения на выбранном языке программирования и вывод числе с заданной точностью.

Задача B. Движение по дуге

Описание робота:

Дифференциальная платформа



D – диаметр колес робота.

L – полуколея робота, расстояние от геометрического центра робота до центра колеса.

Точка *O* – геометрический центр робота, начало локальной системы координат робота. Точка является серединой оси, соединяющей колеса.

Задание:

Робот должен совершить движение по дуге со скоростью *V* м/с. Необходимо вычислить скорости вращения колес, требуемые для совершения заданного движения.

Система оценки

Баллы за каждую подзадачу начисляются только в случае, если все тесты для этой подзадачи и необходимых подзадач успешно пройдены.

Подзадача	Баллы	Ограничения	Необходимые группы тестов
1	50	$R>0$	-
2	50	Основные ограничения	1

Формат входных данных:

Первая строка содержит дробное положительное число L ($10^{-4} \le L \le 1$) — расстояние от центра робота до центра колеса в метрах. «Половина колеи робота».
Вторая строка содержит дробное число R ($-10^3 \le R \le 10^3$) — радиус поворота робота в метрах. Положительное или нулевое значение означает движение по дуге против часовой стрелки, отрицательное — по часовой стрелке.
Третья строка содержит дробное число $V_{роб}$ ($-100 \le V \le 100$) — скорость движения ($V_{роб}$) центра робота (точки *O*) в метрах в секунду. Положительное число означает движение вперед, отрицательное — назад, нулевое — остановку робота.

Четвертая строка содержит дробное положительное число D ($0.003 \le D \le 1$) — диаметр колеса (D) в метрах.

Формат выходных данных

Выведите два числа, содержащие дробные числа (с точностью до 4 знаков после запятой).

В первой строке скорость вращения левого колеса робота, в оборотах в секунду.

Во второй строке скорость вращения правого колеса робота, в оборотах в секунду.

Решение:

Линейные скорости левого и правого колес вычисляются по формулам:

$V_{лев}=V_{роб}*(R-L)/R$

$V_{прав}=V_{роб}*(R+L)/R$

Связь угловой скорости и числа оборот в секунду выражается формулой:

$w=2*\pi*n$ (где n - количество оборотов колеса в секунду).

Линейная скорость связана с угловой следующим выражением, в котором сразу раскроем угловую скорость:

$V=w*R=2*\pi*n*R=\pi*D*n$ (в этом выражении символом R обозначался радиус колеса, а не радиус поворота робота, следовательно $2R=D$).

В этой формуле искомым для текущей задачи является n. Выразим его и подставим выражения из предыдущих формул:

$n=V/(\pi*D)$

$n_{лев}=V_{лев}/(\pi*D)=V_{роб}*(R-L)/(\pi*D*R)$

$n_{прав}=V_{прав}/(\pi*D)=V_{роб}*(R+L)/(\pi*D*R).$

Остается лишь реализовать программное вычисление искомого значения на выбранном языке программирования и вывод числе с заданной точностью.

Задача C. Лабиринт. Правило правой руки

Имя входного файла:	стандартный ввод
Имя выходного файла:	стандартный вывод
Ограничение по времени:	10 секунд
Ограничение по памяти:	256 мегабайт

Описание робота:

Робот представлен абстрактной точкой в лабиринте. Лабиринт состоит из клеток, через которые робот может проехать (.), либо не может проехать (#).

Задание:

Робот проходит лабиринт. Робот может перемещаться между клетками за N секунд. При перемещении он попадает точно в центр клетки. Робот поворачивает внутри клетки на 90 градусов по часовой стрелке (вправо) за M секунд. Лабиринт описывается с помощью символов . и #, где . — это свободная клетка, а # — стена. Гарантируется, что в лабиринте нет блоков с нулями размером 2×2 или больше и робот не может "зациклить"маршрут при прохождении лабиринта.

Требуется вычислить, за какое время робот переместится из стартовой в финишную позицию?

Алгоритм движения робота по "правилу правой руки"

Робот проходит лабиринт по "правилу правой руки повторяя в цикле следующие действия:

1. Проверяет, есть ли стенка справа от робота:
   - Если стенки справа нет, то робот поворачивается направо и делает "шаг вперёд— перемещается на одну клетку.
2. Если справа есть стенка, то проверяет, есть ли стенка впереди:
   - Если стенки впереди нет, то робот делает "шаг вперёд".
3. Если проверки показали, что справа и впереди есть стенки, то робот поворачивает налево.

Правила перемещения робота:

1. Робот может перемещаться между соседними клетками по направлению движения робота за F секунд. При перемещении он попадает точно в центр клетки.
2. Робот может поворачиваться на 90 градусов влево или вправо внутри клетки за R секунд. Поворот изменяет направление движения робота.
3. Робот всегда начинает движение в направлении, указанном в стартовой позиции (например, вправо).
4. Робот не может перемещаться через стены (#).

Формат входных данных

Первая строка содержит два целых числа: H и W — высоту и ширину лабиринта ($5 \le H, W \le 500$).

Вторая строка содержит два целых числа: F и R — время передвижения по направлению движения робота и время поворота ($0 \le F, R \le 10^9$).

Следующие H строк содержат по W символов, описывающих лабиринт:

- . — свободная клетка,
- # — стена,
- S — стартовая позиция робота,
- F — финишная позиция робота.

В последней строке содержится одно целое число d — направление робота ($1 \le d \le 4$).

Описание направления робота:

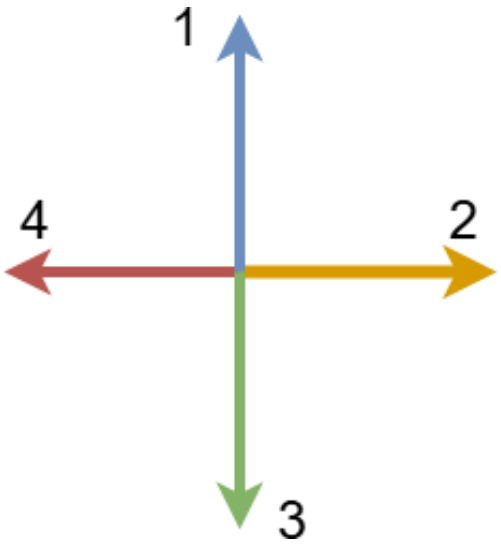
1 — направление вверх,

2 — направление вправо,

3 — направление вниз,

4 — направление влево.

Иллюстрация:



Формат выходных данных

Выведите одно целое число — время, за которое робот сможет добраться из стартовой позиции в финишную.

Система оценки

В задаче 20 тестов, каждый тест оценивается независимо от других в 5 баллов

Примеры

стандартный ввод	стандартный вывод
8 8 1 5 ....# .#.##### .#S....# .#####.# .#...#.# .#.#F#.# .#.#...# ######## 1	31
15 15 1 5 .#............. .#.#####.###### .#.#...#...#... .#.#.#.###.###. ...#.#...#.#... .###.###.#.#.## .#...#.#.#.#... .#.###.#.#.###. .#.#.......#... .#.#########.## S#...#.......#. F###.#.#######. .#.#.#.#.....#. .#.#.#.#.###.#. ...#.....#..... 4	642

Решение:

1 Условие задачи

Робот перемещается по лабиринту, состоящему из клеток двух типов:

- Свободные клетки (обозначены символом .)
- Стены (обозначены символом #)

Стартовая и финишная позиции обозначены символами S и F соответственно. Робот управляется по "правилу правой руки":

1. Если справа нет стены, он поворачивает направо и делает шаг вперёд
2. Если справа стена, но впереди свободно, он делает шаг вперёд
3. Если и справа, и впереди стены, он поворачивает налево

Параметры движения:

- Перемещение на одну клетку занимает F секунд
- Поворот на 90° занимает R секунд
- Начальное направление задаётся числом d:
  - 1 — вверх
  - 2 — вправо
  - 3 — вниз
  - 4 — влево

Требуется вычислить время, за которое робот доберётся от старта до финиша.

2 Алгоритм решения

2.1 Инициализация

1. Считывание размеров лабиринта $H \times W$
2. Считывание времени перемещения F и поворота R
3. Чтение самого лабиринта и определение стартовой и финишной позиций
4. Преобразование начального направления d в индекс (0 — вверх, 1 — вправо, 2 — вниз, 3 — влево)

2.2 Движение робота

Пока текущая позиция робота не совпадает с финишной:

1. Определить направление справа
2. Если справа нет стены:
   - Повернуть направо (увеличить время на R)
   - Сделать шаг вперёд (увеличить время на F)
   - Обновить позицию
3. Иначе если впереди нет стены:
   - Сделать шаг вперёд (увеличить время на F)
   - Обновить позицию
4. Иначе:
   - Повернуть налево (увеличить время на R)

2.3 Завершение

Когда робот достигнет финишной позиции, вывести общее затраченное время.

Задача D. Лабиринт 2

Имя входного файла:	стандартный ввод
Имя выходного файла:	стандартный вывод
Ограничение по времени:	10 секунд
Ограничение по памяти:	256 мегабайт

Описание робота:

Робот представлен абстрактной точкой в лабиринте. Лабиринт состоит из клеток, через которые робот может проехать (.), либо не может проехать (#).

Задание:

Робот проходит лабиринт. Робот может перемещаться между клетками за N секунд. При перемещении он попадает точно в центр клетки. Робот поворачивает внутри клетки на 90 градусов по часовой стрелке (вправо) за M секунд. Лабиринт описывается с помощью символов . и #, где . — это свободная клетка, а # — стена. За какое минимальное время робот переместится из стартовой в финишную позицию?

Правила перемещения робота:

1. Робот может перемещаться между соседними клетками по направлению движения робота за F секунд. При перемещении он попадает точно в центр клетки.
2. Робот может поворачиваться на 90 градусов вправо внутри клетки за R секунд. Поворот из- меняет направление движения робота.
3. Робот всегда начинает движение в направлении, указанном в стартовой позиции (например, вправо).
4. Робот не может перемещаться через стены (#).

Формат входных данных

Первая строка содержит два целых числа: H и W — высоту и ширину лабиринта ($5 \le H, W \le 500$).

Вторая строка содержит два целых числа: F и R — время передвижения по направлению движения робота и время поворота ($0 \le F, R \le 10^9$).

Следующие H строк содержат по W символов, описывающих лабиринт:

- . — свободная клетка,
- # — стена,
- S — стартовая позиция робота,
- F — финишная позиция робота.

В последней строке содержится одно целое число d — направление робота ($1 \le d \le 4$).

Описание направления робота:

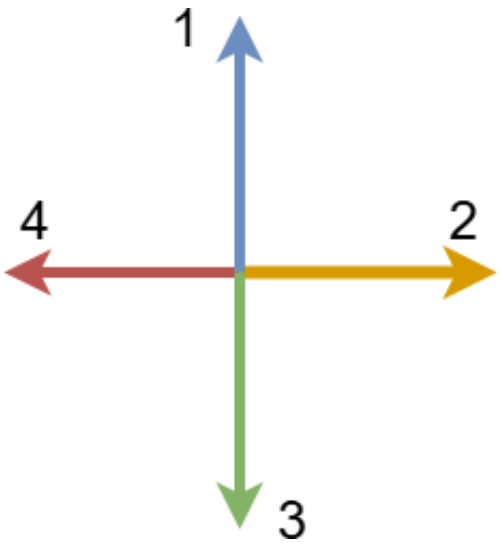
1 — направление вверх,

2 — направление вправо,

3 — направление вниз,

4 — направление влево.

Иллюстрация:



Формат выходных данных

Выведите одно целое число — минимальное время, за которое робот сможет добраться из стартовой позиции в финишную.

Система оценки

Баллы за каждую подзадачу начисляются только в случае, если все тесты для этой подзадачи и необходимых подзадач успешно пройдены.

Подзадача	Баллы	Ограничения	Необходимые группы
1	20	$n, m \leq 10$	—
2	20	Существует только единственный путь от старта до финиша	—
3	30	$F = R$	—
4	30	Основные ограничения	1–3

Примеры

стандартный ввод	стандартный вывод
8 8 1 5 ....# .#.##### .#S....# .#####.# .#...#.# .#.#.#.# .#.#F..# ######## 4	30
8 8 1 5 #F..#... #S....## #.#.#... #.#####. #.#..... #.###.#. #.....#. ######## 2	16

Решение:

1 Условие задачи

Робот перемещается по лабиринту, состоящему из клеток. Клетки могут быть:

- Свободными (.)
- Стенами (#)
- Стартовой позицией (S)
- Финишной позицией (F)

Робот начинает движение из стартовой позиции в заданном направлении и должен достичь финишной позиции за минимальное время.

1.1 Правила перемещения

1. Перемещение вперед занимает F секунд
2. Поворот на 90° занимает R секунд
3. Направления кодируются числами:
   - 1 — вверх
   - 2 — вправо

- 3 — вниз
- 4 — влево

2 Идея решения

Задача сводится к поиску кратчайшего пути в графе, где вершины — это состояния робота, задаваемые:

- Координатами (x, y)
- Текущим направлением dir

Для нахождения кратчайшего пути используется алгоритм Дейкстры, так как веса рёбер (время перемещения или поворота) неотрицательные.

3 Ключевые моменты

- **Состояние робота** описывается тройкой (x, y, dir)
- **Переходы между состояниями**:
  - Перемещение вперед (если клетка свободна) — время $+F$, направление не меняется
  - Поворот — время $+k \cdot R$ (k — количество поворотов), направление меняется
- **Инициализация**: Начальное состояние — стартовая позиция с заданным направлением, время $= 0$
- **Завершение**: Минимальное время среди всех направлений в финишной позиции

4 Алгоритм

1. Инициализация:
   - Массив $dist[x][y][dir]$ (изначально $+\infty$)
   - Очередь с приоритетом для обработки состояний
2. Обработка состояний:
   - Извлечение состояния с минимальным временем
   - Генерация новых состояний:
     - Для каждого возможного поворота
     - Для перемещения вперед (если возможно)
3. Выбор минимального времени в финишной позиции

5 Заключение

Задача решается сведением к поиску кратчайшего пути в графе состояний. Алгоритм Дейкстры эффективно находит оптимальное решение благодаря корректной обработке всех возможных переходов между состояниями робота.

Задача E. Артем и лифт

Имя входного файла:	стандартный ввод
Имя выходного файла:	стандартный вывод
Ограничение по времени:	10 секунд
Ограничение по памяти:	256 мегабайт

Артем построил здание с N этажами. В здании есть лифт, который останавливается на каждом этаже.

На каждом этаже установлена только одна кнопка для управления лифтом — либо «вверх , либо «вниз . Это означает, что с каждого этажа можно двигаться только в одном направлении. Если S_i равно **U**, то на i-м этаже установлена только кнопка «вверх , и можно двигаться только вверх; если S_i равно **D**, то на i-м этаже установлена только кнопка «вниз , и можно двигаться только вниз.

Жильцам приходится добираться до своих этажей через другие этажи, если это необходимо. Требуется найти сумму минимального количества поездок на лифте для всех возможных пар этажей (i, j), где i — начальный этаж, а j — конечный этаж.

Формат входных данных

В первой строке задана строка S ($1 \le |S| \le 10^5$), состоящая из символов **U** и **D**. Каждый символ строки соответствует направлению кнопки на соответствующем этаже.

Формат выходных данных

Выведите сумму следующих чисел для всех упорядоченных пар этажей (i, j): минимальное количество раз, которое нужно воспользоваться лифтом, чтобы добраться с i-го этажа на j-й этаж.

Система оценки

Баллы за каждую подзадачу начисляются только в случае, если все тесты для этой подзадачи и необходимых подзадач успешно пройдены.

Подзадача	Баллы	Ограничения	Необходимые подзадачи	Информация о проверке
1	25	$\|S\| \le 10$	—	первая ошибка
2	25	$\|S\| \le 100$	1	первая ошибка
3	25	$\|S\| \le 10^3$	1, 2	первая ошибка
4	25	Нет дополнительных ограничений	1 — 3	первая ошибка

Примеры

стандартный ввод	стандартный вывод
UDD	7
UUUUUD	40

Замечание

Пояснение к первому примеру:

- $(1; 2)$ — требуется 1 поездка.

- $(1; 3)$ — требуется 1 поездка.
- $(2; 1)$ — требуется 1 поездка.
- $(2; 3)$ — требуется 2 поездки.
- $(3; 2)$ — требуется 1 поездка.
- $(3; 1)$ — требуется 1 поездка.

Решение:

На каждом этаже i лифт имеет направление:

- Если направление на этаже i — 'D', то:
  - Для любого $j < i$ можно попасть с этажа i на этаж j за **1 шаг** (прямой спуск).
  - Для любого $j > i$ можно попасть с этажа i на этаж j за **2 шага**:
    1. Сначала спуститься на самый нижний этаж (так как лифт едет только вниз).
    2. Затем перейти на нужный этаж j (так как с нижнего этажа можно подниматься вверх).
- Если направление на этаже i — 'U', то:
  - Для любого $j > i$ можно попасть с этажа i на этаж j за **1 шаг** (прямой подъём).
  - Для любого $j < i$ можно попасть с этажа i на этаж j за **2 шага**:
    1. Сначала подняться на самый верхний этаж (так как лифт едет только вверх).
    2. Затем перейти на нужный этаж j (так как с верхнего этажа можно спускаться вниз).

Для каждого этажа i можно вычислить сумму расстояний до всех остальных этажей j за **константное время** ($O(1)$), используя предварительно вычисленные префиксные суммы или другие оптимизации. Таким образом, общая сложность алгоритма составляет $O(N)$, где N — количество этажей.

Задача F. Движение по линии

В симуляторе TRIK Studio подготовить управляющую программу, перемещающую робота вдоль линии и останавливающую его на четвертом встреченном перекрестке.

Управляющая программа проверяется на нескольких полях, имеющих разную конфигурацию поворотов, прямых участков и перекрестков. За каждое поле начисляется до 10% баллов. Поле засчитывается при совпадении следующих условий:

- центр робота находится в зоне 300x300 мм, центр которой совпадает с центром четвертого по ходу движения робота перекрестка;
- скорости обоих моторов робота равны нулю.

Описание робота:

Дифференциальная платформа. Робот оснащен четырьмя датчиками отраженного света, направленными вниз:

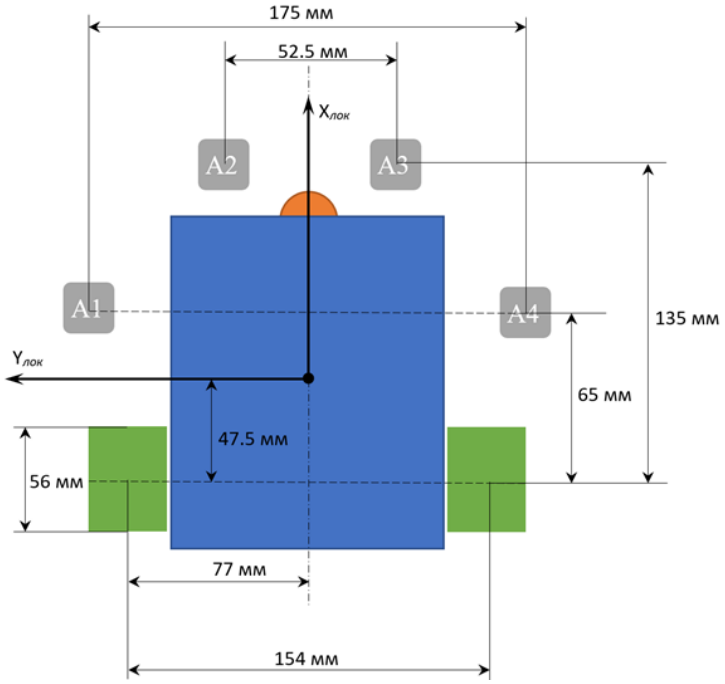
К порту A1 подключен левый боковой датчик.

К порту A2 подключен левый передний датчик.

К порту A3 подключен правый передний датчик.

К порту A4 подключен правый боковой датчик.

Размеры модели в симуляторе TRIK Studio представлены на рисунке.



Левый мотор робота подключен к порту M4, правый мотор - к порту M3.

Энкодеры робота возвращают угол поворота колес в градусах. Положительные числа обозначают вращение колеса, обеспечивающего движение робота вперед, отрицательные - назад.

Полигон:

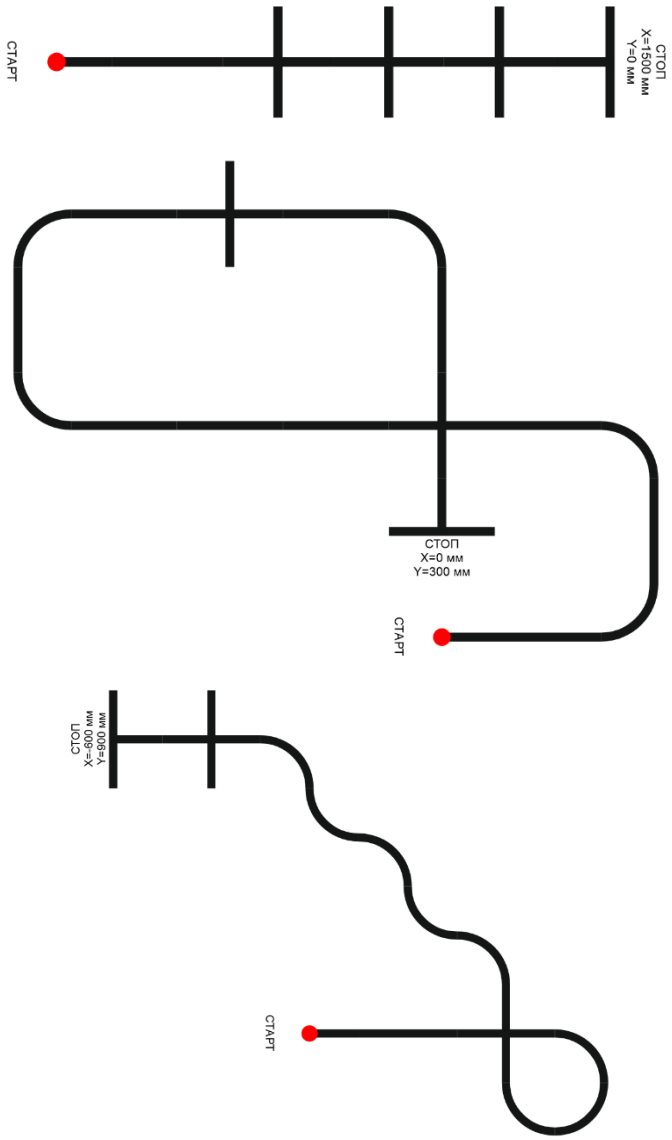
Общий размер полигона составляет 4 х 4 метра.

Стартовая секция и стартовое положение робота - точно в центре полигона, в направлении вправо.

Ширина линии не менее 25 мм.

Гарантируется, что на старте линия расположена между датчиками A2 и A3. Гарантируется, что первые 300 мм линии - прямой участок. Далее следуют повороты с радиусом не менее 150 мм. Гарантируется, что боковые линии на перекрестках отстоят от основной линии не менее чем на 125 мм и имеют ширину не менее 25 мм.

Примеры полей:



Примеры заданий и тестовый проект:

В приложенном архиве находятся следующие фалы:

- Test_field_1_exercise.qrs / Test_field_2_exercise.qrs / Test_field_3_exercise.qrs - файлы-упражнения с настроенными тренировочными полями;
- task_jun_j.py - файл-программа с исходным кодом участника.

Файлы-упражнения могут использоваться для отладки программ. После отладки код программы необходимо перенести в файл task_jun_j.py и отправить в качестве решения задачи.

Решение

Движение по линии может быть реализовано с помощью регулятора. В нем ошибка («невязка») вычисляется как разница в показаниях датчиков на портах A2 и A3. Длительный подбор коэффициентов регулятора можно компенсировать снижением базовой скорости робота, в этом случае даже не идеально подобранные коэффициенты будут работать.

Датчики на портах A1 и A4 можно использовать для определения перекрестков – когда сумма их показаний больше определенного порога можно сделать вывод о том, что робот находится на перекрестке (оба датчика видят черную линию, по условиям задания это возможно только на перекрестках).

После обнаружения черных линий под обоими датчиками необходимо преодолеть перекресток, и только после этого «посчитать» его. Если засчитать перекресток сразу же при обнаружении, не успев съехать с него, то робот сразу же обнаружит второй, третий и последующие перекрестки. Преодолевать перекресток можно либо двигаясь по линии на заданное расстояние или заданное время. Второй вариант проще в реализации. Для ознакомления движение по линии на заданное расстояние можно посмотреть в решении задачи J этой же возрастной группы.

Важно при всем движении по линии не останавливаться на перекрестках, то есть не допускать одновременного обнуления скоростей на обоих моторах. Если же требуется остановиться, то достаточно оставить по 1% мощности на моторах, чтобы проверяющая система не засчитала остановку. Такой малой мощности будет недостаточно для движения робота, но завершение попытки не будет зафиксировано.

Итоговый код программы, реализующей движение робота по линии до четвертого перекрестка, может быть следующим:

```
1. import sys
2. import time
3. import random
4. import math
5.
6. class Program():
7.   __interpretation_started_timestamp__ = time.time() * 1000
8.
9.   pi = 3.141592653589793
10.
11.        def execMain(self):
12.          Diam=0.056 #диаметр колес робота, в метрах
13.          L=0.077 #полуколея робота, в метрах
14.          X=0.0 #координата X робота, в метрах
15.          Y=0.0 #координата Y робота, в метрах
16.          #на старте обе координаты робота нулевые
17.          #во время движения рекомендуется обновлять эти переменные
18.
19.          #добавьте свой код ниже
20.          kf=0.9
21.          M=0
22.          kp=1.5
23.          ki=0
24.          kd=0
25.          err=0
26.          errold=0
```

```
I=0
n=0
leftSpd = 0
rightSpd= 0
Nleft_old=0
Nright_old=0

while n<4:
  s1 = brick.sensor("A1").read()
  s2 = brick.sensor("A2").read()
  s3 = brick.sensor("A3").read()
  s4 = brick.sensor("A4").read()
  s1f = s1
  s4f = s4
  while (s1+s4)<50:
    s1 = brick.sensor("A1").read()
    s2 = brick.sensor("A2").read()
    s3 = brick.sensor("A3").read()
    s4 = brick.sensor("A4").read()
    s1f = kf*s1f+(1-kf)*s1
    s4f = kf*s4f+(1-kf)*s4
    err = s2 - s3
    P=kp*err
    I=I+ki*err
    D=kd*(err-errold)
    U=P+I+D
    leftSpd = 50-U
    rightSpd= 50+U
    brick.motor("M3").setPower(rightSpd)
    brick.motor("M4").setPower(leftSpd)
    errold = err
    Nleft=brick.encoder("M4").read()
    Nright=brick.encoder("M3").read()
    dSleft = math.pi * Diam *(Nleft - Nleft_old)/360
    dSright = math.pi * Diam *(Nright - Nright_old)/360
    X = X+(dSright+dSleft)/2*math.cos( M + (dSright-dSleft)/(4*L) )
    Y = Y+(dSright+dSleft)/2*math.sin( M + (dSright-dSleft)/(4*L) )
    M=M+(dSright-dSleft)/(2*L)
    Nleft_old = Nleft
    Nright_old = Nright
    script.wait(10)
  n=n+1
  startTime = script.time()
  brick.playTone(200, 300)
  dt = 0
  while dt<800:
    s1 = brick.sensor("A1").read()
    s2 = brick.sensor("A2").read()
    s3 = brick.sensor("A3").read()
    s4 = brick.sensor("A4").read()
    s1f = kf*s1f+(1-kf)*s1
    s4f = kf*s4f+(1-kf)*s4
    err = s2 - s3
    P=kp*err
    I=I+ki*err
    D=kd*(err-errold)
    U=P+I+D
    leftSpd = 40-U
    rightSpd= 40+U
    brick.motor("M3").setPower(rightSpd)
    brick.motor("M4").setPower(leftSpd)
    errold = err
    Nleft=brick.encoder("M4").read()
    Nright=brick.encoder("M3").read()
    dSleft = math.pi * Diam *(Nleft - Nleft_old)/360
    dSright = math.pi * Diam *(Nright - Nright_old)/360
    X = X+(dSright+dSleft)/2*math.cos( M + (dSright-dSleft)/(4*L) )
    Y = Y+(dSright+dSleft)/2*math.sin( M + (dSright-dSleft)/(4*L) )
    M=M+(dSright-dSleft)/(2*L)
    Nleft_old = Nleft
    Nright_old = Nright
    script.wait(10)
```

```
            dt=script.time()-startTime


        brick.motor("M3").powerOff()
        brick.motor("M4").powerOff()

        script.wait(10)
        brick.stop()
        return

    def main():
      program = Program()
      program.execMain()

    if __name__ == '__main__':
      main()
```

Задача G. Заезд в клетку

В симуляторе TRIK Studio подготовить управляющую программу, перемещающую робота в клетку с заданными координатами.

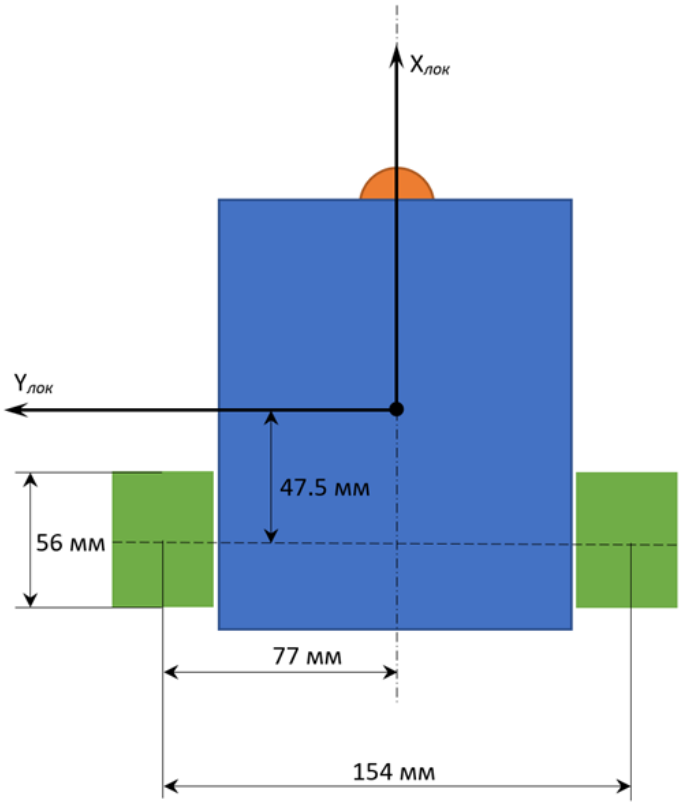
Управляющая программа проверяется на нескольких полях, имеющих разную конфигурацию точек. За каждое поле начисляется до 10% баллов. Поле засчитывается при совпадении следующих условий:

- центр робота находится в клетке с заданными координатами;
- скорости обоих моторов робота равны нулю.

Описание робота:

Дифференциальная платформа.

Размеры модели в симуляторе TRIK Studio представлены на рисунке.



Левый мотор робота подключен к порту M4, правый мотор - к порту M3.

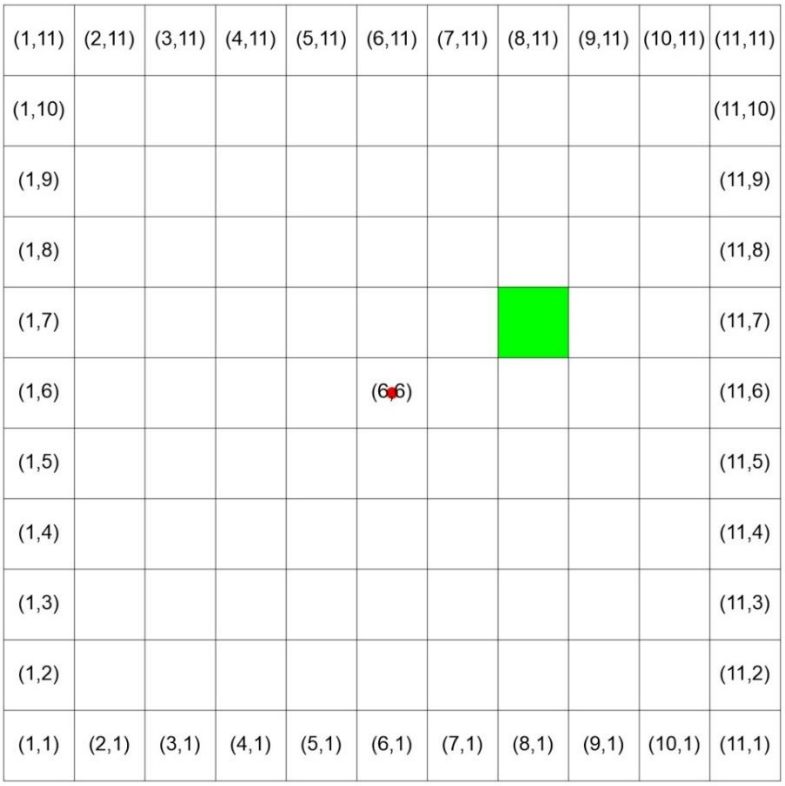
Энкодеры робота возвращают угол поворота колес в градусах. Положительные числа обозначают вращение колеса, обеспечивающего движение робота вперед, отрицательные - назад.

Полигон:

Общий размер полигона составляет 4 x 4 метра. Полигон визуально разделен на клетки размером 300 x 300 мм. Клетки имеют координаты по осям X и Y - их порядковые номера. Всего полигон разделен на 11 клеток по оси X и на 11 клеток по оси Y.

Робот начинает движение в центре клетки с координатами (6, 6) - ровно в середине полигона. Робот направлен вдоль оси OX в положительном направлении (визуально отображается как направление в правую сторону).

Пример полигона:



Примеры заданий и тестовый проект:

В приложенном архиве находятся следующие фалы:

- Test_field_exercise.qrs - файл-упражнение с настроенным тренировочным полем;
- input.txt - файл со входными данными для программы;
- task_jun_l.py - файл-программа с исходным кодом участника.

Файлы-упражнения могут использоваться для отладки программ. После отладки код программы необходимо перенести в файл task_jun_l.py и отправить в качестве решения задачи.

Данные:

Параметры движения содержатся в файле input.txt

В первой строке файла указано целое число от 1 до 11: координата X клетки для посещения. Во второй строке указано целое число от 1 до 11: координата Y клетки для посещения.

Решение

Задача может быть решена разными способами. Один из самых простых – поворот робота к заданной клетке и проезд заданного расстояния по прямой. В условиях задачи не регламентируется направление робота в клетке, поэтому данное решение допустимо.

Вычислить угол можно с помощью обратных тригонометрических функций, наиболее подходящей из которых является арктангенс. Предварительно вычисляются координаты центра клетки относительно центра робота по X и Y (deltaX и deltaY). На этом этапе важно соблюдать знаки, чтобы при вычислении угла не упустить информацию о квадранте. Далее эти deltaX и deltaY передаются как параметры в функцию math.atan2, которая возвращает угол между нулевым направлением (в котором робот ориентирован на старте) и направлением на центр клетки.

Далее требуется вычислить расстояние от центра робота до центра клетки. Сделать это можно по теореме Пифагора: катеты – уже известные deltaX и deltaY, требуется вычислить гипотенузу.

Совершая движение роботу важно синхронизировать моторы, так как в симуляторе включена физика. Без синхронизации робот может двигаться не по прямой линии, а по дуге и «промахнуться мимо клетки». Обратите внимание в итоговом коде программы на функцию SyncMoveDist, которая реализует синхронное движение. В ней настроен регулятор, за ошибку в котором принимается разница в показаниях энкодеров моторов.

Итоговый код программы может быть следующим:

```
import sys
import time
import random
import math

class Program():
  __interpretation_started_timestamp__ = time.time() * 1000

  pi = 3.141592653589793
      Diam=0.056 #диаметр колес робота, в метрах
      L=0.077 #полуколея робота, в метрах
      X=0.0 #координата X робота, в метрах
      Y=0.0 #координата Y робота, в метрах
      orient = 0.0 #ориентация робота, в градусах или радианах
      #на старте все координаты робота нулевые
      #нулевая ориентация означает направление вдоль оси OX
      #во время движения рекомендуется обновлять эти переменные

      encL=0
      encR=0

      def execMain(self):
        #добавьте свой код ниже

        kf=0.9
        M=0
        kp=1.5
        ki=0
        kd=0
        err=0
        errold=0
        I=0
        n=0
        leftSpd = 0
        rightSpd= 0
        Nleft_old=0
        Nright_old=0

        lines = script.readAll("solutions/input.txt")
        zoneX=float(lines[0])
        zoneY=float(lines[1])
```

```
42.
43.           pointX, pointY = self.GetZoneCoordinate(zoneX, zoneY)
44.           self.SyncMoveDist(0.0475)
45.           self.RotateToPoint(pointX, pointY)
46.           dist = self.GetDistFromPoints(self.X, self.Y, pointX, pointY)
47.           self.SyncMoveDist(dist)
48.
49.           brick.stop()
50.           return
51.
52.         def RotateToPoint(self, pointX, pointY):
53.           deltaX = pointX - self.X
54.           deltaY = pointY - self.Y
55.           angleRad = math.atan2(deltaY, deltaX)-self.orient
56.           if angleRad<0:
57.             angleRad=2*math.pi+angleRad
58.           angleDeg = angleRad*180/math.pi
59.           startEncL = brick.encoder("M4").read()
60.           startEncR = brick.encoder("M3").read()
61.           pathAngle = 0
62.           betta = (360*angleRad*self.L)/(math.pi*self.Diam)
63.           baseSPD=15
64.           while pathAngle < math.fabs(betta):
65.             pathL=brick.encoder("M4").read()-startEncL
66.             pathR=brick.encoder("M3").read()-startEncR
67.             pathAngle = ( math.fabs(pathL) + math.fabs(pathR) )/2
68.             err = pathR + pathL
69.             P=1.0*err
70.             U=P
71.             brick.motor("M4").setPower(-baseSPD-U)
72.             brick.motor("M3").setPower(baseSPD-U)
73.           brick.motor("M4").powerOff()
74.           brick.motor("M3").powerOff()
75.
76.         def SyncMoveDist(self, dist):
77.           startEncL = brick.encoder("M4").read()
78.           startEncR = brick.encoder("M3").read()
79.           pathAngle = 0
80.           alpha = (360*dist)/(math.pi*self.Diam)
81.           baseSPD=25
82.           while pathAngle < math.fabs(alpha):
83.             pathL=brick.encoder("M4").read()-startEncL
84.             pathR=brick.encoder("M3").read()-startEncR
85.             pathAngle = (pathL+pathR)/2
86.             err = pathL-pathR
87.             P=1.0*err
88.             U=P
89.             brick.motor("M4").setPower(baseSPD-U)
90.             brick.motor("M3").setPower(baseSPD+U)
91.           brick.motor("M4").powerOff()
92.           brick.motor("M3").powerOff()
93.
94.         def GetDistFromPoints(self, firstPointX, firstPointY, secondPointX,
    secondPointY):
95.           dist = math.sqrt( (secondPointX-firstPointX)**2 + (secondPointY-
    firstPointY)**2 )
96.           return dist
97.
98.         def GetZoneCoordinate(self, zoneX, zoneY):
99.           pointX = (zoneX-6)*0.3
100.          pointY = (zoneY-6)*0.3
101.          return pointX, pointY
102.
103.      def main():
104.        program = Program()
105.        program.execMain()
106.
107.      if __name__ == '__main__':
108.        main()
```

Задача H. Штрих-код

В симуляторе TRIK Studio подготовить управляющую программу, перемещающую робота на заданное расстояние и считывающую штрих-код, представленный в виде линий на полигоне. При этом белая линия кодирует значение 0, черная линия кодирует значение 1.

Управляющая программа проверяется на нескольких полях, имеющих разную конфигурацию линий. За каждое поле начисляется до 10% баллов. Поле засчитывается при совпадении следующих условий:

3 балла начисляются при совпадении условий

- центр робота находится в зоне финиша;
- скорости обоих моторов робота равны нулю.

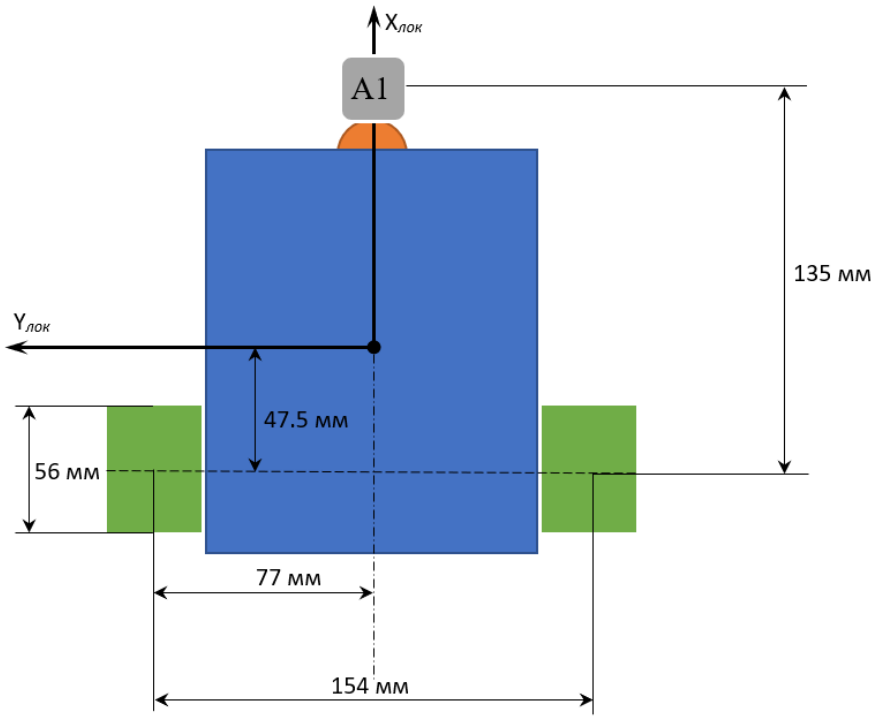
7 баллов начисляются при совпадении условий:

- перед завершением программы робот вывел на экран не менее одного сообщения;
- последнее выведенное на экран сообщение содержит целое число;
- последнее выведенное на экран число совпадает с заданным штрих-кодом в двоичной системе счисления.

Описание робота:

Дифференциальная платформа. Робот оснащен датчиком отраженного света, направленным вниз, подключенным к порту A1.

Размеры модели в симуляторе TRIK Studio представлены на рисунке.



Левый мотор робота подключен к порту M4, правый мотор - к порту M3.

Энкодеры робота возвращают угол поворота колес в градусах. Положительные числа обозначают вращение колеса, обеспечивающего движение робота вперед, отрицательные - назад.

Полигон:

Общий размер полигона составляет 2 х 3 метра.

Робот начинает движение в стартовой точке (на рисунке обозначена красным), в направлении слева направо.

По ходу движения робот проезжает над 12 линиями штрих-кода. Гарантируется, что первая и последняя линии черного цвета. Ширина каждой линии составляет 100 мм. Линия может иметь белый или черный цвет. Первая линия штрих-кода начинается на расстоянии 200 мм от геометрического центра робота (середины оси, соединяющей колеса робота).

Зона финиша расположена за последней линий штрих-кода: на расстоянии 1400 мм от геометрического центра робота. Зона финиша имеет ширину 1000 мм.

Пример полигона:

Код: 100101011101

Примеры заданий и тестовый проект:

В приложенном архиве находятся следующие фалы:

- Test_field_1_exercise.qrs / Test_field_2_exercise.qrs / Test_field_3_exercise.qrs / Test_field_4_exercise.qrs - файлы-упражнения с настроенными тренировочными полями;
- task_jun_m.py - файл-программа с исходным кодом участника.

Файлы-упражнения могут использоваться для отладки программ. После отладки код программы необходимо перенести в файл task_jun_m.py и отправить в качестве решения задачи.

Решение

Роботу для движения не дана линия, поэтому необходимо реализовать прямолинейное движение с синхронизацией моторов. Пример такой реализации можно посмотреть в задаче G этой же возрастной группы.

По условию задачи важно не останавливать робота при считывании линии. Для этого достаточно либо не понижать скорость обоих моторов до 0, оставив хотя бы по 1%, либо считывать код на ходу. Это можно реализовать с помощью отслеживания пройденного расстояния и при превышении пройденного пути на очередной линии производить считывание, не замедляя моторы. Обратите внимание на строки 59-60 приведенной ниже программы, в них как раз устанавливается мощность в 1%.

Пройденное расстояние можно отслеживать по среднему арифметическому показаний энкодеров. Важно корректно выполнить перевод показаний энкодеров в метры или миллиметры.

Проверку линии и считывание одного бита кода удобнее реализовать с помощью функции и вызывать ее при необходимости. В примере ниже это функция CheckLine. Обратите внимание, что черная линия дает бОльшие показания датчика робота.

Обратите внимание на вывод кода на экран робота. Необходимо демонстрировать его именно на экране, а не в консоли.

```
1. import sys
2. import time
3. import random
4. import math
5.
6. class Program():
7.   __interpretation_started_timestamp__ = time.time() * 1000
8.
9.   pi = 3.141592653589793
10.        Diam=0.056 #диаметр колес робота, в метрах
11.        L=0.077 #полуколея робота, в метрах
12.        code = []
13.
14.        def execMain(self):
15.          #добавьте свой код ниже
16.          self.SyncMoveDist(0.250-0.135)
17.          for i in range(0, 12):
18.            val = self.CheckLine()
19.            self.code.append(val)
20.            self.SyncMoveDist(0.100)
21.          mess = ''
22.          for bit in self.code:
23.            mess+=str(bit)
24.          print( mess )
25.          brick.display().clear()
26.          brick.display().addLabel(mess, 1, 1, 20)
27.          brick.display().redraw()
28.          self.SyncMoveDist(0.300)
29.          brick.motor("M4").powerOff()
30.          brick.motor("M3").powerOff()
31.
32.          script.wait(10)
33.          brick.stop()
34.          return
35.
36.        def CheckLine(self):
37.          bit = 0
38.          s1 = brick.sensor("A1").read()
39.          if s1>50:
40.            bit=1
41.          return bit
```

```
42.
43.
44.         def SyncMoveDist(self, dist):
45.           startEncL = brick.encoder("M4").read()
46.           startEncR = brick.encoder("M3").read()
47.           pathAngle = 0
48.           alpha = (360*dist)/(math.pi*self.Diam)
49.           baseSPD=35
50.           while pathAngle < math.fabs(alpha):
51.             pathL=brick.encoder("M4").read()-startEncL
52.             pathR=brick.encoder("M3").read()-startEncR
53.             pathAngle = (pathL+pathR)/2
54.             err = pathL-pathR
55.             P=1.0*err
56.             U=P
57.             brick.motor("M4").setPower(baseSPD-U)
58.             brick.motor("M3").setPower(baseSPD+U)
59.           brick.motor("M4").setPower(1)
60.           brick.motor("M3").setPower(1)
61.
62.       def main():
63.         program = Program()
64.         program.execMain()
65.
66.       if __name__ == '__main__':
67.         main()
```

Задача I. Считывание кода по стенкам

В симуляторе TRIK Studio подготовить управляющую программу, перемещающую робота на заданное расстояние и считывающую штрих-код, представленный в виде стенок на полигоне. При этом близко расположенная стенка кодирует значение 1, далеко расположенная стенка кодирует значение 0.

Управляющая программа проверяется на нескольких полях, имеющих разную конфигурацию стенок. За каждое поле начисляется до 10% баллов. Поле засчитывается при совпадении следующих условий:

3 балла начисляются при совпадении условий

- центр робота находится в зоне финиша;
- скорости обоих моторов робота равны нулю.

7 баллов начисляются при совпадении условий:

- перед завершением программы робот вывел на экран не менее одного сообщения;
- последнее выведенное на экран сообщение содержит целое число;
- последнее выведенное на экран число совпадает с заданным штрих-кодом в двоичной системе счисления.

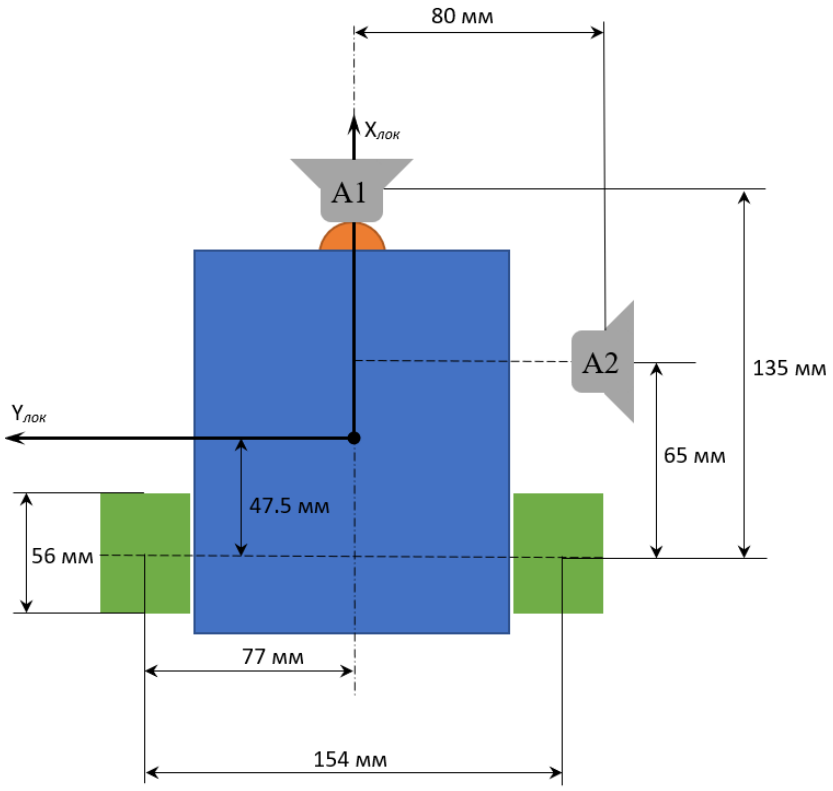
Описание робота:

Дифференциальная платформа. Робот оснащен двумя датчиками расстояния:

К порту A1 подключен датчик, направленный вперед.

К порту A2 подключен датчик, направленный вправо.

Размеры модели в симуляторе TRIK Studio представлены на рисунке.



Левый мотор робота подключен к порту M4, правый мотор - к порту M3.

Энкодеры робота возвращают угол поворота колес в градусах. Положительные числа обозначают вращение колеса, обеспечивающего движение робота вперед, отрицательные - назад.

Полигон:

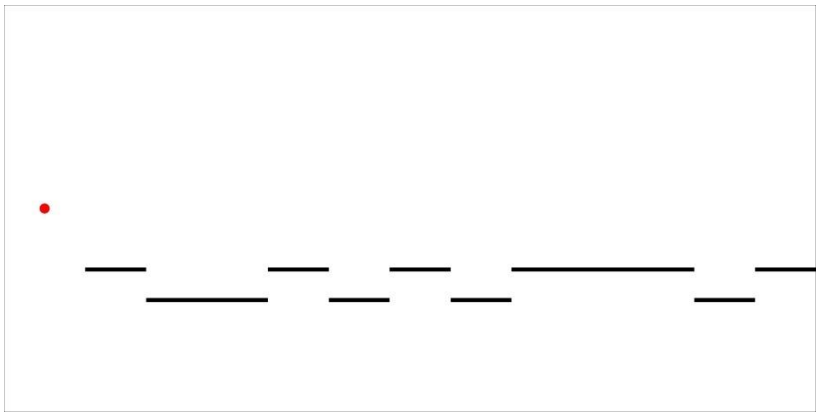
Общий размер полигона составляет 2 х 5 метра.

Робот начинает движение в стартовой точке (на рисунке обозначена красным), в направлении слева направо.

По ходу движения робот проезжает мимо 10 стенок штрих-кода (справа от робота по ходу его движения). Ширина каждой стенки составляет 300 мм. Стенка может быть расположена на расстоянии 300 или 450 мм от стартового вектора движения робота. Первая стенка штрих-кода начинается на расстоянии 200 мм от геометрического центра робота.

Зона финиша расположена за последней линий штрих-кода: на расстоянии 3800 мм от геометрического центра робота. Зона финиша имеет ширину 1000 мм.

Пример полигона:



Код: 100101011101

Примеры заданий и тестовый проект:

В приложенном архиве находятся следующие фалы:

- Test_field_1_exercise.qrs / Test_field_2_exercise.qrs / Test_field_3_exercise.qrs / Test_field_4_exercise.qrs - файлы-упражнения с настроенными тренировочными полями;
- task_jun_n.py - файл-программа с исходным кодом участника.

Файлы-упражнения могут использоваться для отладки программ. После отладки код программы необходимо перенести в файл task_jun_n.py и отправить в качестве решения задачи.

Решение

Задача очень похожа на задачу H этой же возрастной группы. Отличие лишь в считывании кода – не по датчику освещения, а по датчику расстояния. Программа с итоговым решением отличается от задачи H только в функции CheckLine и расстоянии проезда между битами кода.

```
1. import sys
2. import time
3. import random
4. import math
5.
6. class Program():
7.   __interpretation_started_timestamp__ = time.time() * 1000
8.
9.   pi = 3.141592653589793
10.        Diam=0.056 #диаметр колес робота, в метрах
11.        L=0.077 #полуколея робота, в метрах
12.        code = []
13.
14.        def execMain(self):
15.          #добавьте свой код ниже
16.
17.          brick.display().clear()
18.          self.SyncMoveDist(0.2+0.15-0.135)
19.          for i in range(0, 12):
20.            val = self.CheckDist()
21.            self.code.append(val)
22.            self.SyncMoveDist(0.3)
23.          mess = ''
24.          for bit in self.code:
25.            mess+=str(bit)
```

```
26.           print( mess )
27.           brick.display().addLabel(mess, 1, 1, 20)
28.           brick.display().redraw()
29.           self.SyncMoveDist(0.3)
30.           brick.motor("M4").powerOff()
31.           brick.motor("M3").powerOff()
32.
33.           script.wait(10)
34.           brick.stop()
35.           return
36.
37.         def CheckDist(self):
38.           bit = 0
39.           s2 = brick.sensor("A2").read()
40.           if s2<(40-10):
41.             bit=1
42.           return bit
43.
44.
45.         def SyncMoveDist(self, dist):
46.           startEncL = brick.encoder("M4").read()
47.           startEncR = brick.encoder("M3").read()
48.           pathAngle = 0
49.           alpha = (360*dist)/(math.pi*self.Diam)
50.           baseSPD=35
51.           while pathAngle < math.fabs(alpha):
52.             pathL=brick.encoder("M4").read()-startEncL
53.             pathR=brick.encoder("M3").read()-startEncR
54.             pathAngle = (pathL+pathR)/2
55.             err = pathL-pathR
56.             P=1.0*err
57.             U=P
58.             brick.motor("M4").setPower(baseSPD-U)
59.             brick.motor("M3").setPower(baseSPD+U)
60.           brick.motor("M4").setPower(1)
61.           brick.motor("M3").setPower(1)
62.
63.       def main():
64.         program = Program()
65.         program.execMain()
66.
67.       if __name__ == '__main__':
68.         main()
```

Задача J. Перекрестки

В симуляторе TRIK Studio подготовить управляющую программу, перемещающую робота по линиям, образующим сетку, в заданные перекрестки.

Управляющая программа проверяется на одном поле, но с разными последовательностями посещения перекрестков. За каждое поле начисляется до 10% баллов. При проверке на каждом поле начисляется по 2 балла за каждый посещенный перекресток при совпадении следующих условий:

- робот переместился в перекресток, указанный ему для посещения, то есть любая точка проекции робота находится над центром указанного перекрестка;
- робот переместился в указанный перекресток в верном порядке, то есть предыдущий указанный перекресток был засчитан, либо это первый указанный по порядку посещения перекресток;
- скорости обоих моторов робота равны нулю.

Описание робота:

Дифференциальная платформа. Робот оснащен четырьмя датчиками отраженного света, направленными вниз:

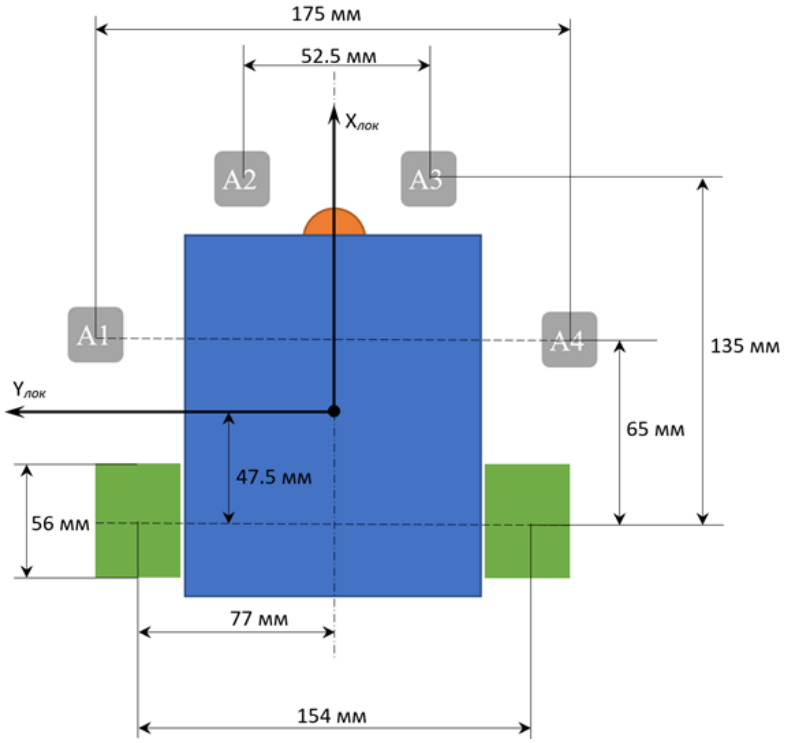
К порту A1 подключен левый боковой датчик.

К порту A2 подключен левый передний датчик.

К порту A3 подключен правый передний датчик.

К порту A4 подключен правый боковой датчик.

Размеры модели в симуляторе TRIK Studio представлены на рисунке.



Левый мотор робота подключен к порту M4, правый мотор - к порту M3.

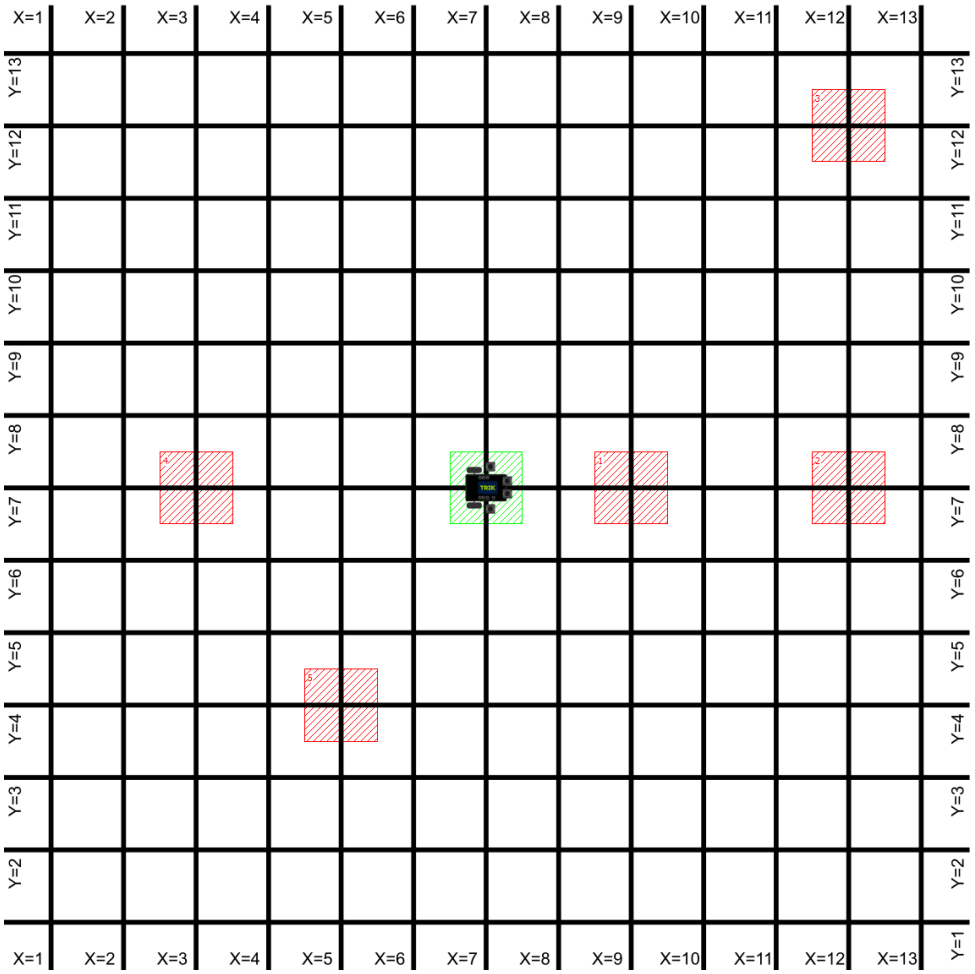
Энкодеры робота возвращают угол поворота колес в градусах. Положительные числа обозначают вращение колеса, обеспечивающего движение робота вперед, отрицательные - назад.

Полигон:

Общий размер полигона составляет 4 x 4 метра. На полигоне нанесены прямые линии, образующие на пересечениях перекрестки. Толщина линий - 20 мм, расстояние между центрами линий - 300 мм. Перекрестки имеют координаты по осям X и Y - их порядковые номера. Всего полигон разделен на 13 перекрестков по оси X и на 13 перекрестков по оси Y.

Робот начинает движение в центральном перекрестке с координатами (7, 7) - ровно в середине полигона. Робот направлен вдоль оси OX в положительном направлении (визуально отображается как направление в правую сторону).

Пример полигона:



Пример файла input.txt для примера полигона:

9 7

12 7

12 12

3 7

5 4

Примеры заданий и тестовый проект:

В приложенном архиве находятся следующие фалы:

- Test_field_exercise.qrs - файл-упражнение с настроенным тренировочным полем;
- input.txt - файл со входными данными для программы;
- task_sen_l.py - файл-программа с исходным кодом участника.

Файлы-упражнения могут использоваться для отладки программ. После отладки код программы необходимо перенести в файл task_sen_l.py и отправить в качестве решения задачи.

Данные:

Параметры движения содержатся в файле input.txt

В первой строке файла указаны два целых числа от 1 до 13, разделенных пробелом: координаты X и Y первого перекрестка для посещения. В следующих строках указаны аналогичные пары чисел, обозначающие координаты следующих перекрестков для посещения. Ограничения и значения чисел аналогичны таким же из первой строки. Количество строк соответствует количеству перекрестков для посещения.

Решение

Декомпозируем задачу на составляющие. В самом простом случае роботу необходимо уметь выполнять следующие отдельные действия:

1. двигаться по линии;
2. определять и подсчитывать перекрестки;
3. поворачивать в заданном направлении;
4. вычислять разницу между текущими координатами (текущим перекрестком) и заданными.

Повторяя эти действия в цикле, можно перемещаться по заданным точкам.

Движение по линии и определение перекрестков были реализовано ранее в задаче F этой возрастной группы. Для удобства их использования можно оформить эти части кода в виде функций.

Дополнительно удобно реализовать проезд по лини на заданное расстояние. В этом случае можно после определения перекрестка датчиками делать «доезд» (на расстояние между датчиками и колесами) и оказываться колесами ровно на перекрестке. Такая установка робота обеспечивает удобное стартовое положение для поворотов, робот попадет точно датчиками на линию.

Поворот в самом простом случае можно реализовать только в одну сторону (например, против часовой стрелки) и повторять до тех пор, пока робот не окажется в верном направлении.

Итоговый код программы может выглядеть следующим образом:

```
import sys
import time
import random
import math

class Program():
  __interpretation_started_timestamp__ = time.time() * 1000

  pi = 3.141592653589793
      Diam=0.056 #диаметр колес робота, в метрах
      L=0.077 #полуколея робота, в метрах
      X=7 #координата X робота, в метрах
      Y=7 #координата Y робота, в метрах
      M=0 #ориентация робота в пространстве, в градусах или радианах на усмотрение участника
      #на старте все координаты робота обозначают его стартовое положение
      #во время движения рекомендуется обновлять эти переменные
      kf=0.9
      kp=1.0
      ki=0
      kd=0
      errold=0
      I=0
      s1f=0
      s2f=0
      s3f=0
      s4f=0

      def execMain(self):
        #добавьте свой код ниже
        crossroads = script.readAll("solutions/input.txt")
        for crossroad in crossroads:
          cross = [int(i) for i in crossroad.split()]
          crossX=cross[0]
          crossY=cross[1]
          self.MoveToCrosroads(crossX, crossY)
        brick.motor("M3").powerOff()
        brick.motor("M4").powerOff()

        script.wait(10)
        brick.stop()
        return

      def Turn(self, orient):
        turnNum=orient-self.M
        turnNum=turnNum%4
        n=0
        while n<turnNum:
          self.RotateToLine()
          n=n+1
        self.M = orient

      def RotateToLine(self):
        brick.motor("M3").setPower(25)
        brick.motor("M4").setPower(-25)
        script.wait(1000)
        s3 = brick.sensor("A3").read()
        while s3<30:
          s3 = brick.sensor("A3").read()
        brick.motor("M4").powerOff()
        brick.motor("M3").powerOff()

      def MoveToCrosroads(self, endCrossX, endCrossY):
        deltaX = endCrossX - self.X
        deltaY = endCrossY - self.Y
        if deltaX>0:
          self.Turn(0)
          self.Crossroads( deltaX )
        if deltaX<0:
          self.Turn(2)
          self.Crossroads( math.fabs(deltaX) )
        if deltaY>0:
```

```
72.             self.Turn(1)
73.             self.Crossroads( deltaY )
74.           if deltaY < 0:
75.             self.Turn(3)
76.             self.Crossroads( math.fabs(deltaY) )
77.           pass
78.
79.         def LineTrace(self):
80.           s2 = brick.sensor("A2").read()
81.           s3 = brick.sensor("A3").read()
82.           err = s2 - s3
83.           P=self.kp*err
84.           self.I=self.I+self.ki*err
85.           D=self.kd*(err-self.errold)
86.           U=P+self.I+D
87.           leftSpd = 50-U
88.           rightSpd= 50+U
89.           brick.motor("M3").setPower(rightSpd)
90.           brick.motor("M4").setPower(leftSpd)
91.           self.errold = err
92.           script.wait(10)
93.           pass
94.
95.         def LineToCrossroad(self):
96.           s1 = brick.sensor("A1").read()
97.           s4 = brick.sensor("A4").read()
98.           while (s1+s4)<60:
99.             s1 = brick.sensor("A1").read()
100.            s4 = brick.sensor("A4").read()
101.            self.LineTrace()
102.          brick.motor("M4").powerOff()
103.          brick.motor("M3").powerOff()
104.
105.        def LineToDist(self, dist):
106.          startEncL = brick.encoder("M4").read()
107.          startEncR = brick.encoder("M3").read()
108.          pathAngle=0
109.          alpha = (360*dist)/(math.pi*self.Diam)
110.          while pathAngle < math.fabs(alpha):
111.            pathL=brick.encoder("M4").read()-startEncL
112.            pathR=brick.encoder("M3").read()-startEncR
113.            pathAngle = (pathL+pathR)/2
114.            self.LineTrace()
115.          brick.motor("M4").powerOff()
116.          brick.motor("M3").powerOff()
117.
118.        def Crossroads(self, crossNum):
119.          n=0
120.          while n<crossNum:
121.            self.LineToCrossroad()
122.            self.LineToDist(0.0475)
123.            n=n+1
124.            if self.M==0:
125.              self.X = self.X + 1
126.            if self.M==1:
127.              self.Y = self.Y + 1
128.            if self.M==2:
129.              self.X = self.X - 1
130.            if self.M==3:
131.              self.Y = self.Y - 1
132.            brick.playTone(1000, 100)
133.
134.      def main():
135.        program = Program()
136.        program.execMain()
137.
138.      if __name__ == '__main__':
139.        main()
```

Старшая возрастная группа

Задача A. Закон управления

Описание робота

Робот представлен абстрактной точкой, способной двигаться вперед и назад по следующим правилам:

- максимальная скорость движения робота вперед или назад V_{max} м/с;
- максимальное ускорение робота при разгоне a_1 м/с^2;
- максимальное ускорение робота при торможении a_2 м/с^2.

Задание

Роботу необходимо преодолеть расстояние S метров. Робот должен завершать движение (в точке на расстоянии S от старта) с нулевой скоростью. За какое минимальное время он может выполнить это движение?

Система оценки

В данной задаче 20 тестов, каждый тест оценивается в 5 баллов.

Формат входных данных:

Первая строка содержит дробное число V_{max} ($0.0001 \le V_{max} \le 10^3$) — максимальную скорость движения робота в м/с.
Вторая строка содержит дробное число a_1 ($0.0001 \le a_1 \le 10^3$) — максимальное ускорение робота при разгоне в м/с^2.
Третья строка содержит дробное число a_2 ($0.0001 \le a_2 \le 10^3$) — максимальное ускорение робота при торможении в м/с^2.
Четвёртая строка содержит дробное число S ($0.0001 \le S \le 10^4$) — расстояние, которое требуется преодолеть роботу, в метрах.

Формат выходных данных

Выведите одно дробное число (с точностью до 4 знаков после запятой) — минимально необходимое время для совершения движения в секундах.

Решение:

Вспомним из школьного курса физики равномерное и равноускоренное движение. Путь, проделанный при равномерном движении:

$S=S_0+V*t$

Путь, проделанный при равноускоренном движении:

$S=S_0+V_0*t+a*t^2/2$

В нашем случае S_0 равен 0, так как робот начинает движение в "нулевой точке".

Разделим движение, совершаемое роботом, на три возможных отрезка:

- разгон - движение на этом участке равноускоренное, скорость изменяется от нуля до скорости установившегося движения с ускорением a_1, следовательно: $S_p=a_1*t_p^2/2$, при этом из определения ускорения $t_p=(V_{max}-V_0)/a_1=V_{max}/a_1$.

- установившееся движение - движение с установившейся скоростью V_y; для достижения наименьшего времени общего движения эта скорость должна быть максимально возможной, то есть V_{max}, следовательно: $S_y=V_{max}*t_y$.
-
- торможение - движение на этом участке равнозамедленное, скорость изменяется от установившейся (то есть V_{max}) до нуля с отрицательным ускорением a_2, следовательно: $S_m = V_{max}*t_m-a_2*t_m^2/2$, при этом из определения ускорения $t_p=(V_0-V_{max})/-a_2=V_{max}/a_2$.

Исходя из указанных выше выражений вычислим суммарный путь робота при разгоне и торможении:

$S_{pm} = S_p+S_m = a_1*t_p^2/2+V_{max}*t_m-a_2*t_m^2/2 = a_1/2*(V_{max}/a_1)^2+V_{max}*V_{max}/a_2-a_2/2*(V_{max}/a_2)^2 = V_{max}^2/(2*a_1)+V_{max}^2/a_2-V_{max}^2/(2*a_2) = V_{max}^2/(2*a_1)+V_{max}^2/(2*a_2)$.

Далее возможны три варианта (учитываем, что по условию задачи все расстояния строго положительны):

- $S > S_{pm}$ - общий путь робота превышает путь при разгоне до максимально возможной скорости и торможения от нее, то есть робот успевает разогнаться до максимума, двигаться с максимальной скоростью и тормозить от нее до нуля. В этом случае
  $S_y = S-S_{pm}$
  $t_y = (S-S_{pm})/V_{max}$
  $t_{общ} = t_p+t_y+t_m$.
- $S = S_{mp}$ - общий путь робота при равен пути при разгоне и торможении до максимально возможной скорости и торможения от нее, то есть робот успевает только разогнаться до максимальной скорости и тормозить от нее до нуля, но не успевает двигаться равномерно. В этом случае
  $S_y=0$
  $t_y=0$
  $t_{общ}=t_p+t_m$.
- $S < S_{mp}$ - общий путь робота настолько мал, что робот не успевает разогнаться до максимально возможной скорости и ему уже следует тормозить. То есть робот не достигает максимально возможной скорости V_{max}, а достигает лишь какой-то промежуточной скорости, назовем ее $V_н$. В этом случае присутствует только путь при разгоне и торможении, но до скорости $V_н$

$$S = \frac{V_н^2}{2a_1} + \frac{V_н^2}{2a_2} = V_н^2 \frac{a_1 + a_2}{2a_1 a_1}$$

$$V_н = \sqrt{\frac{2a_1 a_2 S}{a_1 + a_2}}$$

$$t_{общ} = \frac{V_н}{a_1} + \frac{V_н}{a_2}.$$

Остается лишь реализовать программное вычисление искомого значения на выбранном языке программирования и вывод числе с заданной точностью. Выбор одного из вариантов вычисления времени реализуется простым сравнением общего пути и предварительно вычисленных путей разгона и торможения.

Задача B. Размер объекта по камере

Имя входного файла:	стандартный ввод
Имя выходного файла:	стандартный вывод
Ограничение по времени:	10 секунд
Ограничение по памяти:	256 мегабайт

Робот имеет камеру со следующей оптической системой:

- W — ширина кадра камеры, в пикселях;
- H — высота кадра камеры, в пикселях;
- F — угол обзора камеры, в градусах.

Задание

Робот обнаружил камерой препятствие. Датчик расстояния определил расстояние до препятствия D. Препятствие занимает на кадре с камеры N пикселей в высоту. Необходимо определить реальный размер препятствия.

Формат входных данных

Первая строка содержит два целых положительных числа W и H ($128 \le W, H \le 4096$) — разрешение камеры по ширине (W) и высоте (H), в пикселях.

Вторая строка содержит дробное положительное число F ($0 < F < 180$) — угол обзора оптики камеры (F) в градусах.

Третья строка содержит целое положительное число D ($5 \le D \le 4000$) — расстояние до препятствия, в миллиметрах.

Четвёртая строка содержит целое положительное число N ($128 \le N \le 4096$) — количество пикселей, занимаемое препятствием на кадре с камеры.

Формат выходных данных

Выведите дробное число (с точностью до 4 знаков после запятой) — высоту препятствия в метрах.

Система оценки

В данной задаче 50 тестов, каждый тест оценивается в 2 балла.

стандартный ввод	стандартный вывод
128 128 90 3500 300	16.40625

Решение:

Алгоритм решения:

1. Преобразование угла обзора в радианы: Угол обзора F задан в градусах, но для математических вычислений (например, для функции ‘math.tan‘) его нужно перевести в радианы. Это делается с помощью функции ‘math.radians‘.
2. Расчёт реальной высоты кадра: Реальная высота кадра H_{real} может быть вычислена по формуле:

$$H_{\text{real}} = 2 \cdot D \cdot \tan\left(\frac{F_{\text{rad}}}{2}\right)$$

Здесь: D — расстояние до препятствия. F_{rad} — угол обзора в радианах.

3. Расчёт реальной высоты препятствия: Реальная высота препятствия H_{object} вычисляется пропорционально количеству пикселей N, которые оно занимает на кадре:

$$H_{\text{object}} = \frac{H_{\text{real}} \cdot N}{H}$$

Здесь: H — высота кадра в пикселях. N — количество пикселей, занимаемое препятствием.

4. Преобразование в метры: поскольку расстояние D задано в миллиметрах, результат нужно перевести в метры, разделив на 1000.
5. Округление результата: Результат выводим с точностью 4 знаков после запятой.

Задача C. Робот и лазеры

Имя входного файла:	стандартный ввод
Имя выходного файла:	стандартный вывод
Ограничение по времени:	10 секунд
Ограничение по памяти:	256 мегабайт

Робот шириной L оборудован двумя лазерными указками синего и красного цвета. Лазерные указки закреплены так, чтобы они пересекались на некотором строго заданном расстоянии от робота. Синяя лазерная указка повернута на угол α относительно корпуса робота, красная — на угол β.

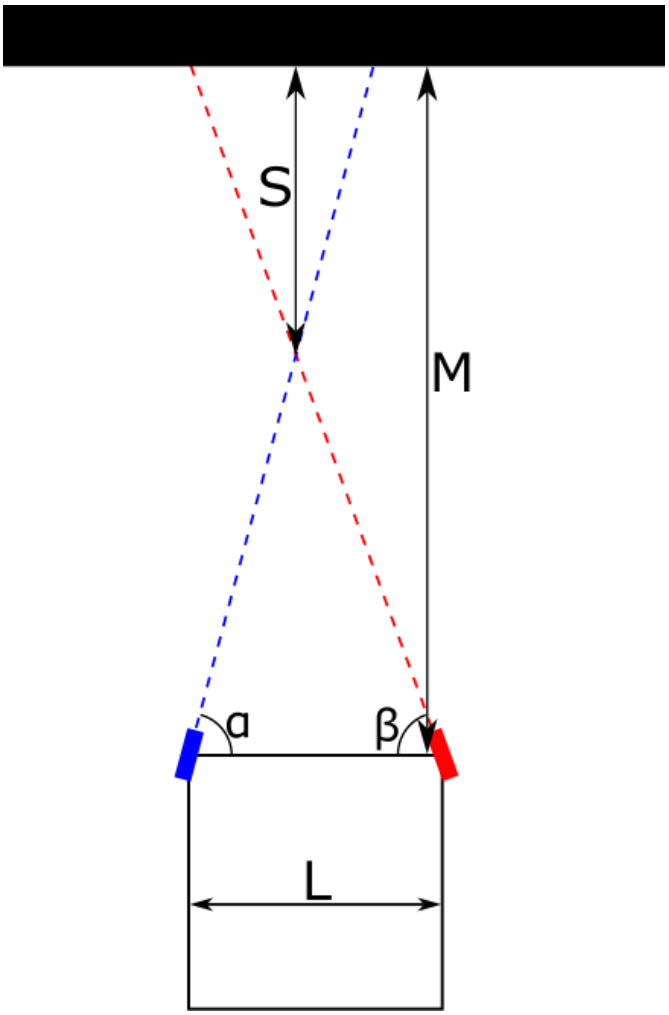


Рис. 1: Схема расположения лазерных указок

Камера робота может отслеживать пятна лазерных указок на поверхности стены/препятствия. При этом:

- Если красное пятно левее синего, то робот находится дальше заданного расстояния от стены.
- Если красное пятно правее синего, то робот находится ближе заданного расстояния от стены.
- Если синее и красное пятна сливаются, то робот находится на заданном расстоянии от стены.

Задание:

Робот стартует на расстоянии M от стены. Ему необходимо остановиться на заданном расстоянии, при котором пятна от лазерных указок сливаются. Гарантируется, что робот стартует на расстоянии от стены большем, чем заданное пересечением лазерных указок.

Необходимо определить, какое расстояние S преодолеет робот до остановки перед стеной.

Формат входных данных

Первая строка содержит целое число L ($1 \le L \le 10^4$) — ширина робота в миллиметрах.

Вторая строка содержит дробное число α ($0.0001 \le \alpha \le 180$) — угол между корпусом робота и лучом синей лазерной указки в градусах.

Третья строка содержит дробное число β ($0.0001 \le \beta \le 180$) — угол между корпусом робота и лучом красной лазерной указки в градусах.

Четвертая строка содержит целое число M ($1 \le M \le 10^5$), которое обозначает расстояние до стены в миллиметрах.

Формат выходных данных

Выведите одно целое число — расстояние S, которое преодолеет робот до остановки перед стеной.

Система оценки

Баллы за каждую подзадачу начисляются только в случае, если все тесты для этой подзадачи и необходимых подзадач успешно пройдены.

Подзадача	Баллы	Ограничения	Необходимые группы тестов
1	25	$\alpha, \beta \in \{30, 60, 90\}$	—
2	25	$\alpha = \beta$	—
3	25	$\alpha, \beta \le 90$	—
4	25	Основные ограничения	1, 2, 3

Пример

стандартный ввод	стандартный вывод
10 30 45 100	96.33975

Решение:

Для решения задачи необходимо определить расстояние S, которое робот должен преодолеть, чтобы оказаться на заданном расстоянии от стены, где пятна от лазерных указок сливаются.

1. Перевод углов в радианы: Углы α и β заданы в градусах, но для математических вычислений удобнее работать с радианами. Поэтому переводим углы в радианы:

$$\alpha_{\text{rad}} = \alpha \cdot \frac{\pi}{180}, \quad \beta_{\text{rad}} = \beta \cdot \frac{\pi}{180}$$

2. Проверка корректности углов: Если сумма углов $\alpha + \beta \ge 180^\circ$, то лучи лазерных указок не пересекутся перед роботом, и задача не имеет решения. В этом случае выводим -1.
3. Расчет расстояния до точки пересечения лучей: Используем тригонометрические соотношения для определения расстояния до точки пересечения лучей. Расстояние D до точки пересечения лучей можно найти по формуле:

$$D = \frac{L \cdot \sin(\alpha) \cdot \sin(\beta)}{\sin(\alpha + \beta)}$$

Здесь L — ширина робота, α и β — углы в радианах.

4. Расчет расстояния S: Робот стартует на расстоянии M от стены, а должен остановиться на расстоянии D. Следовательно, расстояние S, которое робот должен преодолеть, равно:

$$S = M - D$$

5. Вывод результата: выводим значение S с точностью до пяти знаков после запятой

Задача D. Время движения

Имя входного файла:	стандартный ввод
Имя выходного файла:	стандартный вывод
Ограничение по времени:	2.5 секунд
Ограничение по памяти:	256 мегабайт

Описание робота:

Робот представляет собой грузовик, способный двигаться по следующим правилам:

- Максимальная скорость движения робота: v м/с.
- Максимальное ускорение робота при разгоне: a_1 м/с^2.
- Максимальное ускорение робота при торможении: a_2 м/с^2.

Задание:

Роботу необходимо за один раз перевезти максимальное количество деталей из города 1 в город n. Не все дороги могут выдержать вес грузовика с деталями; у каждой дороги есть ограничение на максимальный вес грузовика в граммах. Вес каждой детали составляет 100 грамм. Поскольку это инновационный грузовик, вес деталей не влияет на разгон, торможение и максимальную скорость. Робот может работать только в течение 24 часов, после чего он выключается, и детали не будут доставлены. При этом робот должен останавливаться в каждом городе, через который проезжает, чтобы предотвратить перегрев двигателей.

Робот совершит поездку только один раз. Все дороги являются двусторонними.

Формат входных данных

В первой строке находятся два целых числа n и m, разделенные пробелом, где n ($2 \le n \le 500$) обозначает количество городов, а m ($1 \le m \le 10^5$) — количество двусторонних дорог.

Во второй строке содержатся три дробных числа a_1, a_2 и v, разделенные пробелами, где a_1 и a_2 ($0.0001 \le a_1, a_2 \le 10^3$) обозначают максимальное ускорение робота при разгоне и торможении в м/с^2, а v ($0.0001 \le v \le 10^3$) — максимальную скорость движения робота в м/с.

Следующие m строк описывают дороги. В каждой строке находятся четыре целых числа a_i, b_i, d_i и w_i, разделенные пробелами, где:

- a_i и b_i — номера городов, которые соединяет дорога ($1 \le a_i, b_i \le n$).
- d_i — расстояние в метрах между городами ($d_i \le 10^7$).
- w_i — ограничение на вес автомобиля в граммах ($w_i \le 10^9$).

Между каждой парой городов есть не более одной дороги. Пустой робот-грузовик весит 3 тонны.

Формат выходных данных

Выведите одно целое число — наибольшее количество деталей, которое может доставить робот.

Система оценки

Баллы за каждую подзадачу начисляются только в случае, если все тесты для этой подзадачи и необходимых подзадач успешно пройдены.

Подзадача	Баллы	Ограничения	Необходимые группы тестов
1	30	$n \le 10$	—

2	30	$n \leq 100$	—
3	40	Основные ограничения	1, 2

Примеры

стандартный ввод	стандартный вывод
3 3 197.8795 976.1743 164.0841 1 2 1559.0201 3000220 2 3 3199.8608 3000201 1 3 82.2635 3000099	2
2 1 122.0067 542.1186 344.2097 1 2 495067.1114 3000100	1
3 3 966.6536 763.8393 514.1287 1 3 740035.5261 3000100 1 2 740035.5261 3000200 2 3 213.3404 3000200	2

Решение:

1 Подробный разбор задачи о роботе-грузовике

1.1 Постановка задачи

Робот-грузовик должен перевезти максимальное количество деталей из начального города в конечный за время не более 24 часов. При этом:

- Грузовик имеет физические ограничения: максимальную скорость, ускорение и торможение
- Каждая дорога между городами имеет ограничение по максимальному весу
- Вес грузовика без груза составляет 3 тонны, каждая деталь весит 100 грамм
- Грузовик должен останавливаться в каждом промежуточном городе

1.2 Общая схема решения

Решение задачи состоит из трех основных этапов:

1. Расчет минимального времени прохождения каждой дороги с учетом физических ограничений грузовика
2. Поиск пути с учетом ограничений по весу и времени
3. Определение максимального количества деталей бинарным поиском

1.3 Этап 1: Расчет времени прохождения дороги

Для каждой дороги длиной S метров вычисляем минимальное время прохождения, для этого возьмем решение из задачи «Закон управления.

1.4 Этап 2: Поиск пути с ограничениями

Для заданного количества деталей k (и соответствующего веса $W = 3000000 + 100k$ грамм) ищем путь, удовлетворяющий:

- Вес грузовика на каждой дороге пути ≤ ограничению дороги
- Суммарное время пути ≤ 24 часов

1.4.1 Алгоритм Дейкстры с модификациями

1. Инициализируем массив минимальных времен для каждого города бесконечностью
2. Начальному городу присваиваем время 0
3. Используем очередь с приоритетом для обработки городов
4. Для каждого города рассматриваем все соседние дороги:
   - Пропускаем дороги, где ограничение веса меньше W
   - Если время до соседнего города через текущую дорогу меньше известного - обновляем
5. Продолжаем, пока не обработаем все города или не найдем путь в конечный город

1.5 Этап 3: Поиск максимального количества деталей

Используем бинарный поиск для нахождения максимального k:

- Левая граница: 0 деталей
- Правая граница: максимально возможное количество (из ограничений задачи)
- На каждой итерации:
  - Берем среднее значение *mid*
  - Проверяем, существует ли путь для *mid* деталей (Этап 2)
  - Если существует — двигаем левую границу
  - Если не существует — двигаем правую границу
- Когда границы сойдутся — получаем ответ

1.6 Оптимизации и важные замечания

- Предварительный расчет времени для всех дорог ускоряет решение
- Для точности вычислений используем double (особенно в физических расчетах)
- Остановка Дейкстры при достижении конечного города экономит время
- Учет 24 часов в секундах (86400) упрощает сравнение

1.7 Заключение

Задача решается комбинацией физических расчетов движения, алгоритмов поиска пути на графах и оптимизационного поиска.

Задача E. Artag

Имя входного файла:	стандартный ввод
Имя выходного файла:	стандартный вывод
Ограничение по времени:	5 секунд
Ограничение по памяти:	512 мегабайт

Постановка задачи

Для выполнения задания роботу-доставщику необходимо узнать адрес, куда нужно доставить груз. Адрес имеет 9 бит информации и представлен в виде ArTag метки размером 3×3. Вам необходимо написать программу, определяющую id, на который указывает ArTag метка, увиденная роботом. Изображение представляется на вход программы в виде двумерного массива значений оттенков серого от 0 до 255. Двумерный массив получен с помощью преобразования в градации серого (grayscale).

Справочная информация

ArTag

ArTag метка представляет собой квадратное изображение, которое можно разделить на 25 ячеек. Внешние ячейки всегда имеют чёрный цвет и обозначают границу ArTag метки. Внутренние 9 ячеек (квадрат 3×3 ячейки) содержат код ArTag маркера. Каждый ArTag маркер хранит уникальный целочисленный идентификатор id ($1 \le id \le 383$). Белый цвет ячейки кодирует значение 1 для данного бита, чёрный — 0. Если записать значения ячеек кода ArTag маркера в виде двоичного числа и затем перевести в десятичное, то мы получим соответствующий id маркера. Полный словарь со всеми возможными значениями id маркеров и их соответствий двоичному коду вы можете скачать внизу страницы.

Взвешенный метод преобразования в градации серого (Grayscale)

Взвешенный метод преобразования изображения в градации серого учитывает различную чувствительность человеческого глаза к разным цветам. Чаще всего используется формула, основанная на стандарте **ITU-R BT.601**, которая учитывает вклад каждого цветового канала (красного, зелёного и синего) в яркость:

Формула:

$$\text{gray} = 0.299 \times R + 0.587 \times G + 0.114 \times B$$

Пояснение:

- R, G, B — значения красного, зелёного и синего каналов соответственно.
- Коэффициенты 0.299, 0.587 и 0.114 отражают вклад каждого канала в восприятие яркости:
  - Зелёный канал (G) имеет наибольший вес, так как человеческий глаз наиболее чувствителен к зелёному цвету.
  - Красный (R) и синий (B) каналы имеют меньший вес.

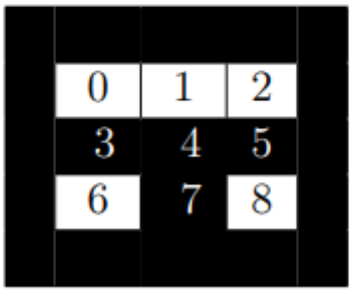
Пример:

Если пиксель имеет значения $R = 100$, $G = 150$, $B = 200$, то:

$$\text{gray} = 0.299 \times 100 + 0.587 \times 150 + 0.114 \times 200 \approx 137.65$$

Округляем до целого числа: 138

Примеры таблиц



Формат входных данных

Первая строка содержит одно целое число t ($1 \le t \le 100$) — номер теста.
Первая строка содержит два целых числа n и m ($1 \le n, m \le 1000$) — размер двумерного массива.
В следующих n строках по m целых чисел от 0 до 255.
В задаче три примера в файле samlpe.zip.

Формат выходных данных

Выведите одно число id.

Система оценки

В задаче 97 тестов: первые 94 теста оцениваются в 1 балл, а последние 3 теста — по 2 балла.

Решение:

Далее приведён пример решения на Python без использования NumPy и OpenCV (будем использовать matplotlib для отладки/отображения текущих результатов, для сдачи в тестовую систему отображение надо будет отключить).

В качестве исходных данных предполагается файл “sample\Test-3.txt” со следующей условной структурой:

Первая строка – количество изображений (например, 1).

Вторая строка – ширина (width) и высота (height) (например, 500 500).

Далее – построчно значения пикселей от 0 до 255, для каждой строки изображения. Всего будет height строк, в каждой строке по width чисел.

```python
def read_image_from_file(filename):
    """
    Считываем данные из файла формата:
      1) номер теста (int)
      2) width height (int int)
      3) height строк по width чисел 0-255
    Возвращаем 2D-массив яркостей (градаций серого) размером [height][width].
    """
    with open(filename, 'r') as f:
        tokens = f.read().split()
    # tokens теперь список всех чисел файла (строки разбиты по пробелам).

    # Первое число — номер теста (например, 1):
    index = 0
    test_n= int(tokens[index])
    index += 1

    # Далее для каждого изображения (у нас обычно 1) читаем width и height:
    width = int(tokens[index])
    index += 1
    height = int(tokens[index])
    index += 1

    # Создаём матрицу под итоговые пиксели в градациях серого
    image = [[0]*width for _ in range(height)]

    # Считываем все пиксели
    # Для удобства и скорости округлим в int сразу
    for row in range(height):
        for col in range(width):

            image[row][col] = int(tokens[index])
            index += 1

    return image
```

На рисунке 1 приведено изображение после чтения из файла

Рисунок 1. Изображение после чтения файла

После чтения данных мы бинаризуем (переводим в чёрно-белое) изображение с порогом 125:

если pixel<125→ 0 (чёрный),

иначе → 255 (белый).

Приведем код функции бинаризации

```
_black_and_white(image, threshold=125): """
Бинаризация изображения (2D-матрица) с порогом threshold. Значения <
threshold -> 0 (чёрный), иначе -> 1 (белый). """
height = len(image)
width = len(image[0]) if height > 0 else 0

bw_image = [[0]*width for _ in range(height)] for row in

range(height):
            for col in range(width):
                if image[row][col] < threshold:
                    bw_image[row][col] = 0
                else:
                bw_image[row][col] = 1 return
bw_image
```

На рисунке 2 приведено изображение после перевода его в черно-белый режим.

Рисунок 2. Изображение Ч/Б

Далее перед нами стоит задача нахождения четырех углов ArTag. В условиях сказано, что ArTag всегда имеет вокруг себя черную рамку, таким образом даже с учетом взгляда на Artag под углом и в перспективе он будет на плоском изображении представлять из себя выпуклый четырехугольник. Ниже приведён простой метод, который находит каждый угол чёрной рамки, основанный на том, что при движении «по диагоналям» от соответствующего угла изображения мы первым делом наткнёмся именно на угол рамки метки (первый чёрный пиксель). Для левого верхнего угла мы начинаем «волной» от точки (0,0) и идём по диагоналям:

Сначала проверяем (0,0). Затем (0,1) и (1,0). Затем (0,2), (1,1), (2,0) и так далее. Как только встретили чёрный пиксель (0) — это и есть искомый угол рамки. Аналогично для правого верхнего угла (от (0, width−1)), левого нижнего (от (height−1,0)), правого нижнего (от (height−1,width−1)).

Ниже приведен пример кода для поиска одного из углов:

```
def find_top_left_corner(bw_image):
    """
    Ищем угол рамки, видимый при движении по диагоналям из (0,0).
    bw_image: 2D-массив, где 0 = чёрный, 255 = белый.
    Возвращает (row, col).
    """
    height = len(bw_image)
    width = len(bw_image[0]) if height > 0 else 0

    # Максимальное возможное значение "суммы индексов" = (height-1) + (width-1)
    max_d = (height - 1) + (width - 1)
```

```
    # Перебираем "слои" (d = i + j)
    for d in range(max_d + 1):
        # Для каждого d перебираем i от 0 до d
        for i in range(d + 1):
            j = d - i
            # Проверяем, не вышли ли за границы
            if i < height and j < width:
                if bw_image[i][j] == 0:
                    return (j, i) # (x, y)
    # Если ничего не нашли (что маловероятно при условии, что угол существует),
    # вернём None или бросим исключение
    return None
```

найдем все четыре угла ArTag метки и сохраним их координаты в один список:

```
def find_corners_diagonal(bw_image):
    """
    Находим все 4 угла рамки, используя диагональные проходы
    из каждого угла изображения.
    Возвращает список [(r_tl, c_tl), (r_tr, c_tr), (r_bl, c_bl), (r_br, c_br)].
    """
    c_tl = find_top_left_corner(bw_image)
    c_tr = find_top_right_corner(bw_image)
    c_bl = find_bottom_left_corner(bw_image)
    c_br = find_bottom_right_corner(bw_image)

    return [c_tl, c_tr, c_bl, c_br]
```

После нахождения углов может их графически отобразить на картинке красными точками (рисунок 3).

Рисунок 3. ArTag после нахождения внешних углов

Теперь зная углы ArTag метки нам необходимо определить значения значащих битов этой метки. Для этого можно построить матрицу гомографии (перспективного преобразования) и перевести найденных четырехугольник в правильный квадрат.

Такой способ более правильный, однако может быть избыточно сложным для школьной аудитории. В связи с этим мы применим более простой способ. Мы знаем что внутри найденного четырехугольника располагается сетка из значений размером 5 на 5 элементов. Найдем центр каждой из таких ячеек в сетке. Используем билинейную интерполяцию. Пусть

$r \in [0,1]$— «доля» по вертикали (от верхней кромки к нижней), а $c \in [0,1]$ — «доля» по горизонтали (от левого края к правому).

Тогда любая точка (x,y) внутри четырёхугольника может быть приблизительно найдена по формуле: P(r,c)=(1−r)*((1−c)*TL+c*TR)+r*((1−c)*BL+c*BR),

где:

TL=(xtl, ytl) — верхний левый угол, TR=(xtr, ytr) — верхний правый, BL=(xbl, ybl) — нижний левый, BR=(xbr, ybr) — нижний правый.

Так как у нас 5×5 ячеек, то по вертикали и горизонтали будет 5 «полос», а значит 6 линий (но мы хотим именно центры ячеек). Поэтому для ячейки в «строке» i и «столбце» j (где i,j $\in$ {0,1,2,3,4}) мы берём:

$r=(i+0.5)/5$, $c=(j+0.5)/5$

Это даёт 25 точек (по одной на каждую ячейку).

Пример реализации функции:

```
def get_sub_cell_centers(corners):
    """
    corners: список из 4 точек (x, y) в порядке:
      0) верхний левый (TL)
      1) верхний правый (TR)
      2) нижний левый (BL)
      3) нижний правый (BR)
    Возвращаем список из 25 точек (x, y) – центров ячеек.
    """
    # Распакуем углы для удобства чтения
    (xtl, ytl) = corners[0]  # Top-Left
    (xtr, ytr) = corners[1]  # Top-Right
    (xbl, ybl) = corners[2]  # Bottom-Left
    (xbr, ybr) = corners[3]  # Bottom-Right

    centers = []
    # Проходим по i=0..4 (строки), j=0..4 (столбцы)
    for i in range(5):
        for j in range(5):
            # r и c – доля от 0 до 1
            r = (i + 0.5) / 5.0
            c = (j + 0.5) / 5.0
```

```
            # Линейная интерполяция по верхней границе: top = TL*(1-c) + TR*c
            top_x = xtl*(1 - c) + xtr*c
            top_y = ytl*(1 - c) + ytr*c

            # Линейная интерполяция по нижней границе: bottom = BL*(1-c) + BR*c
            bot_x = xbl*(1 - c) + xbr*c
            bot_y = ybl*(1 - c) + ybr*c

            # Теперь интерполяция по вертикали между top и bottom
            center_x =int (top_x*(1 - r) + bot_x*r)
            center_y = int ( top_y*(1 - r) + bot_y*r)

            centers.append((center_x, center_y))

    return centers
```

Эти 25 точек (центров) мы можем записать в список и отобразить на исходном изображении синими точками (рисунок 4).

Рисунок 4. Центры значащих ячеек ArTag

Следующим шагом составим матрицу 3*3 значений цветов из основной зоны ArTag, напомни что внешние ячейки не используются для самого кода, а лишь четко обозначают зону, где находится сама метка.

```
def get_3x3_matrix(image, centers_5x5):
    """
    Из 5×5 центров берём внутренние 3×3.
    Возвращаем матрицу (список из 3 списков, по 3 элемента) с битами 0 или 1.

    image[row][col] — яркость (0..255).
    centers_5x5[k] = (x, y)  (учтите, что x=col, y=row).
    """
    mat_3x3 = []
    for i in range(1, 4):  # строки 1..3
```

```
        row_bits = []
        for j in range(1, 4):  # столбцы 1..3
            (x, y) = centers_5x5[i*5 + j]
            # Берем координаты
            pix_val = image[y][x]
            bit = pix_val
            row_bits.append(bit)
        mat_3x3.append(row_bits)
    return mat_3x3
```

Для определения самого кода нам понадобится преобразовать матрицу в линейный массив, так как именно так хранятся значения в словаре aruco_3x3_dict. Для этого напишем отдельную функцию:

```
def flatten_3x3(mat):
    """
    Превращаем 3×3 матрицу в список из 9 элементов (построчно).
    """
    # mat[r][c], r=0..2, c=0..2
    arr = []
    for r in range(3):
        for c in range(3):
            arr.append(mat[r][c])
    return arr
```

Словарь составлен не случайным образом. В нем отсутствуют такие значения, которые при любом из поворотов матрицы 3 на 3 на 90, 180, 270 градусов могли бы дать одинаковые значения. Таким образом вне зависимости от того, с какой стороны робот будем смотреть на ArTag метку он сможет безошибочно определить её уникальный номер. Таким образом нам надо тоже уметь вращать матрицу кратно 90 градусам. Напишем функцию, которая берёт 3×3 матрицу и возвращает её повёрнутую на 90° по часовой стрелке. Для 3×3 это можно сделать формулой:

new[r][c]=old[3−1−c][r]

```
def rotate_3x3_90(mat):
    """
    Поворачивает матрицу mat (3×3) на 90 градусов по часовой стрелке.
    Возвращает новую 3×3 матрицу.
    """
    n = 3
    rotated = [[0]*n for _ in range(n)]
    for r in range(n):
        for c in range(n):
            rotated[r][c] = mat[n-1-c][r]
    return rotated
```

Для определения кода составим все возможные повороты найденной матрицы 3*3, каждый из них преобразует в линейный массив и проверим на наличие в словаре aruco_3x3_dict. Как только

найдем совпадение, остановим перебор вариантов. Пример вращений матрицы приведен на рисунке 5.

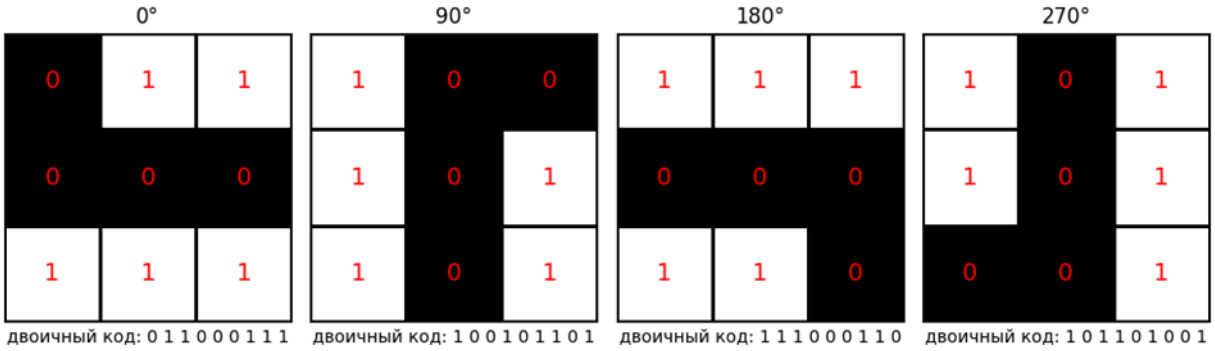


*Рисунок 5. пример вращения матрицы 3*3*

```
def decode_aruco_3x3(mat_3x3, aruco_dict):
    """
    Проверяем 4 поворота (0,90,180,270).
    mat_3x3: матрица 3×3 с битами 0/1.
    aruco_dict: словарь {key_id: [9 бит], ...}.
    Возвращаем key_id, если нашли совпадение, иначе None.
    """
    # 0° (без поворота)
    mat0 = mat_3x3
    arr0 = flatten_3x3(mat0)

    # 90°
    mat90 = rotate_3x3_90(mat0)
    arr90 = flatten_3x3(mat90)

    # 180°
    mat180 = rotate_3x3_90(mat90)
    arr180 = flatten_3x3(mat180)

    # 270°
    mat270 = rotate_3x3_90(mat180)
    arr270 = flatten_3x3(mat270)

    # Сформируем список всех вариантов
    candidates = [arr0, arr90, arr180, arr270]
    #print(candidates)

    # Для каждого кандидата проверяем, есть ли в словаре
    for candidate in candidates:
        for key_id, pattern in aruco_dict.items():
            if candidate == pattern:
                return key_id  # нашли совпадение

    # Если ничего не подошло
    return None
```

-

Итак мы смогли найти сам ArTag на изображении, найти все 9 бит его значащей информации и перебрать варианты поворотами на 90гр для поиска единственно возможного значения. Итоговый код основной программы на базе подготовленных функций будет следующим:

```
def main():
    # 1. Считываем изображение из файла и переводим в градации серого
    filename = 'sample\\Test-3.txt'
    image_gray = read_image_from_file(filename)
    # 2. Применим бинаризацию с порогом 125
    bw_image = to_black_and_white(image_gray, threshold=125)
    # 3. Ищем углы
    corners = find_corners_diagonal(bw_image)

    # 4. Получаем список центров 25 ячеек
    centers_5x5 = get_sub_cell_centers(corners)

    # 5. Получаем 3×3 матрицу (биты)
    mat_3x3 = get_3x3_matrix(bw_image, centers_5x5)

    # 6. Проверяем по словарю
    found_key = decode_aruco_3x3(mat_3x3, aruco_3x3_dict)
    if found_key is not None:
        print("Найден ключ в словаре:", found_key)
    else:
        print("Такого кода в словаре нет.")
```

Задача F. Робот

Имя входного файла:	стандартный ввод
Имя выходного файла:	стандартный вывод
Ограничение по времени:	1 секунда
Ограничение по памяти:	256 мегабайт

Эльвира решила научить своего робота передвигаться по плоскости. Однако для упрощения она ограничила дистанцию, на которую робот может перемещаться за один шаг, значением k. Чтобы робот мог добраться от начальной точки до конечной, Эльвира добавила на плоскость несколько промежуточных точек. Теперь робот может передвигаться только между заданными точками.

Если робот перемещается от точки i с координатами (xi, yi) до точки j с координатами (xj, yj), то он затрачивает (xi — xj)2 + (yi — yj)2 единиц времени.

Формат входных данных

В первой строке заданы числа n и k ($1 \le n \le 5000$, $1 \le k \le 10^9$) .

В следующих n строках заданы координаты точек x_i, y_i ($1 \le x_i, y_i \le 20000$) .

В последней строке заданы два числа s и t ($1 \le s, t \le n$, $s \neq t$) — начальная и конечная точки пути робота.

Формат выходных данных

Выведите единственное число — минимальное время за которое робот доберется из s в t или —1, если робот не может добраться.

Система оценки

Баллы за каждую подзадачу начисляются только в случае, если все тесты для этой подзадачи и необходимых подзадач успешно пройдены.

Подзадача	Баллы	Ограничения	Необходимые подзадачи	Информация о проверке
1	25	$n \le 6$	—	первая ошибка
2	25	$n \le 10$	1	первая ошибка
3	25	$n \le 10^3$	1, 2	первая ошибка
4	25	Нет дополнительных ограничений	1 — 3	первая ошибка

Примеры

стандартный ввод	стандартный вывод
2 32 1 1 5 5 1 2	32
2 31 1 1 5 5 1 2	-1

Решение:

Формулировка задачи в терминах теории графов

Даны точки на плоскости. Необходимо найти минимальное расстояние между двумя заданными точками s и t, при условии, что перемещаться можно только между такими парами точек, для которых квадрат расстояния не превышает некоторого порога k.

Моделирование графом

Преобразуем исходную задачу в задачу поиска пути в графе:

- Каждая точка соответствует **вершине графа**.
- Между двумя вершинами проводится **ребро**, если квадрат расстояния между соответствующими точками $\leq k$.
- Вес ребра равен евклидову расстоянию между точками.

Выбор алгоритма решения

Для нахождения кратчайшего пути в таком графе применим **алгоритм Дейкстры**, поскольку:

- Он эффективно работает с графами с неотрицательными весами рёбер.
- Его временная сложность $O(|E| \log |V|)$ при использовании приоритетной очереди делает его подходящим для многих практических случаев.

Алгоритм решения

1. Построить граф, соединив рёбрами все пары точек (u, v), для которых $||u - v||^2 \leq k$.
2. Назначить веса рёбер равными расстояниям между соответствующими точками.
3. Применить алгоритм Дейкстры для нахождения кратчайшего пути от s до t.

Задача G. Штрих-код

В симуляторе TRIK Studio подготовить управляющую программу, перемещающую робота на заданное расстояние и считывающую штрих-код, представленный в виде линий на полигоне. При этом белая линия кодирует значение 0, черная линия кодирует значение 1.

Управляющая программа проверяется на нескольких полях, имеющих разную конфигурацию линий. За каждое поле начисляется до 10% баллов. Поле засчитывается при совпадении следующих условий:

3 балла начисляются при совпадении условий

- центр робота находится в зоне финиша;
- скорости обоих моторов робота равны нулю.

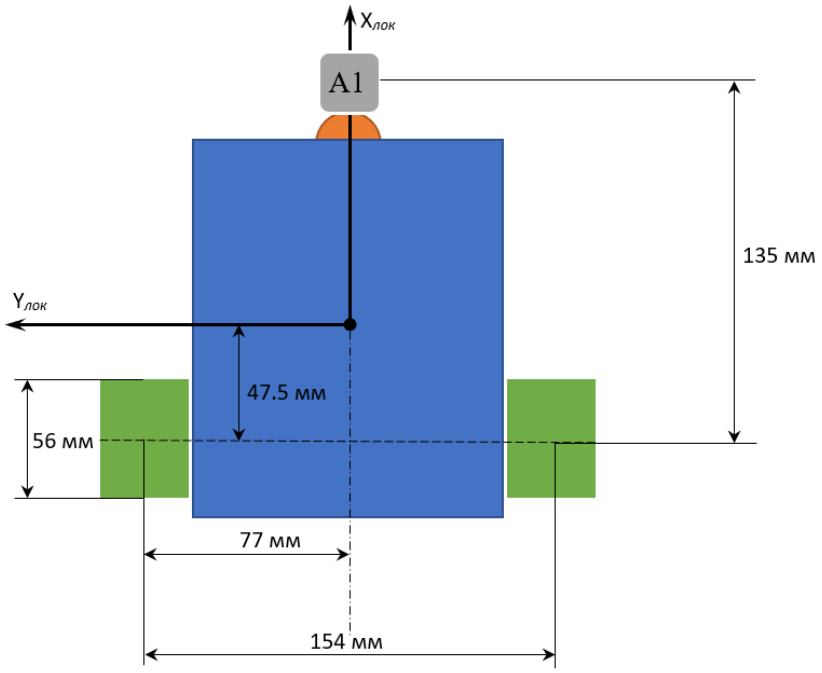
7 баллов начисляются при совпадении условий:

- перед завершением программы робот вывел на экран не менее одного сообщения;
- последнее выведенное на экран сообщение содержит целое число;
- последнее выведенное на экран число совпадает с заданным штрих-кодом в двоичной системе счисления.

Описание робота:

Дифференциальная платформа. Робот оснащен датчиком отраженного света, направленным вниз, подключенным к порту A1.

Размеры модели в симуляторе TRIK Studio представлены на рисунке.



Левый мотор робота подключен к порту M4, правый мотор - к порту M3.

Энкодеры робота возвращают угол поворота колес в градусах. Положительные числа обозначают вращение колеса, обеспечивающего движение робота вперед, отрицательные - назад.

Полигон:

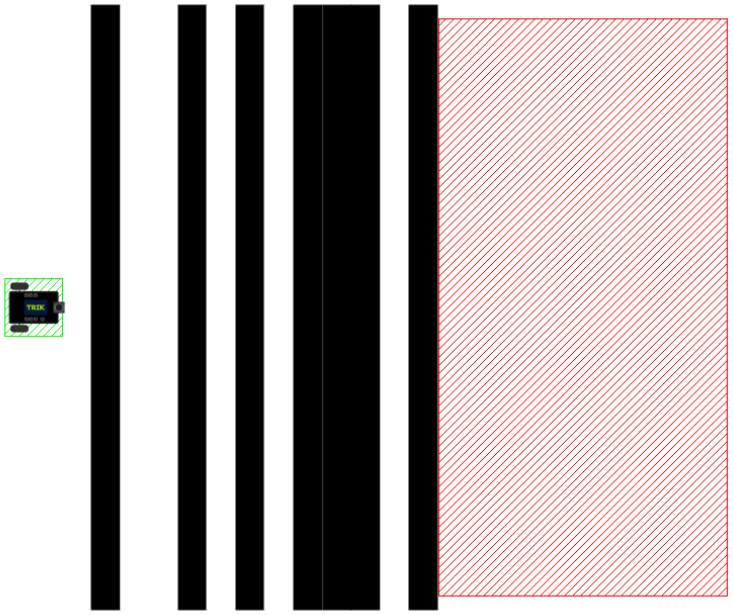
Общий размер полигона составляет 2 х 3 метра.

Робот начинает движение в стартовой точке (на рисунке обозначена красным), в направлении слева направо.

По ходу движения робот проезжает над 12 линиями штрих-кода. Гарантируется, что первая и последняя линии черного цвета. Ширина каждой линии составляет 100 мм. Линия может иметь белый или черный цвет. Первая линия штрих-кода начинается на расстоянии 200 мм от геометрического центра робота (середины оси, соединяющей колеса робота).

Зона финиша расположена за последней линий штрих-кода: на расстоянии 1400 мм от геометрического центра робота. Зона финиша имеет ширину 1000 мм.

Пример полигона:



Код: 100101011101

Примеры заданий и тестовый проект:

В приложенном архиве находятся следующие фалы:

- Test_field_1_exercise.qrs / Test_field_2_exercise.qrs / Test_field_3_exercise.qrs / Test_field_4_exercise.qrs - файлы-упражнения с настроенными тренировочными полями;
- task_jun_m.py - файл-программа с исходным кодом участника.

Файлы-упражнения могут использоваться для отладки программ. После отладки код программы необходимо перенести в файл task_jun_m.py и отправить в качестве решения задачи.

Решение

См. решение задачи H возрастной группы 7-8 классов.

Задача H. Движение по линии

В симуляторе TRIK Studio подготовить управляющую программу, перемещающую робота вдоль линии и останавливающую его на N встреченном перекрестке.

Управляющая программа проверяется на нескольких полях, имеющих разную конфигурацию поворотов, прямых участков и перекрестков. Номер N передается роботу отдельно для каждого поля. За каждое поле начисляется до 10% баллов. Поле засчитывается при совпадении следующих условий:

- центр робота находится в зоне 300x300 мм, центр которой совпадает с центром N-го по ходу движения робота перекрестка;
- скорости обоих моторов робота равны нулю.

Описание робота:

Дифференциальная платформа. Робот оснащен четырьмя датчиками отраженного света, направленными вниз:

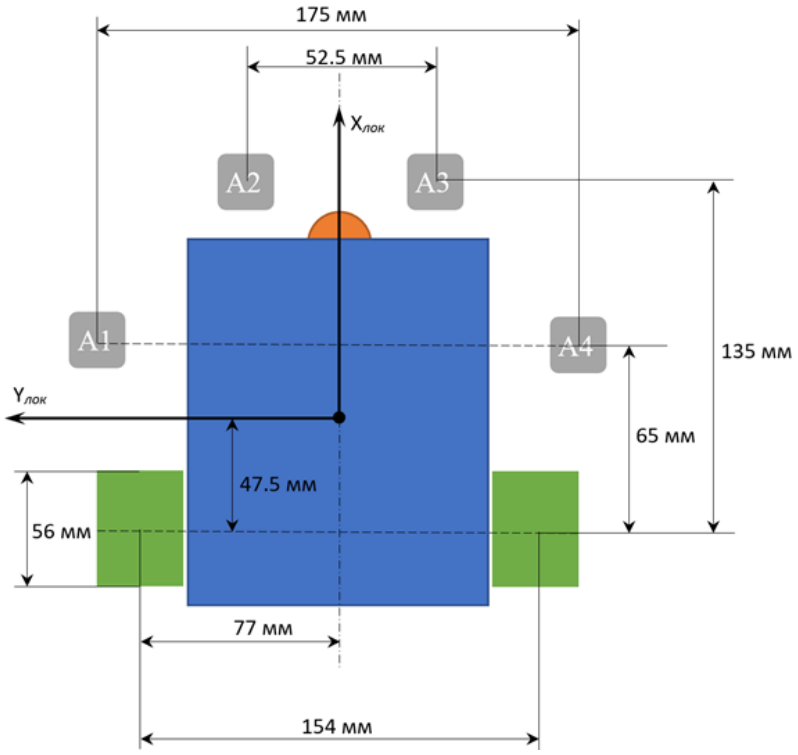
К порту A1 подключен левый боковой датчик.

К порту A2 подключен левый передний датчик.

К порту A3 подключен правый передний датчик.

К порту A4 подключен правый боковой датчик.

Размеры модели в симуляторе TRIK Studio представлены на рисунке.



Левый мотор робота подключен к порту M4, правый мотор - к порту M3.

Энкодеры робота возвращают угол поворота колес в градусах. Положительные числа обозначают вращение колеса, обеспечивающего движение робота вперед, отрицательные - назад.

Полигон:

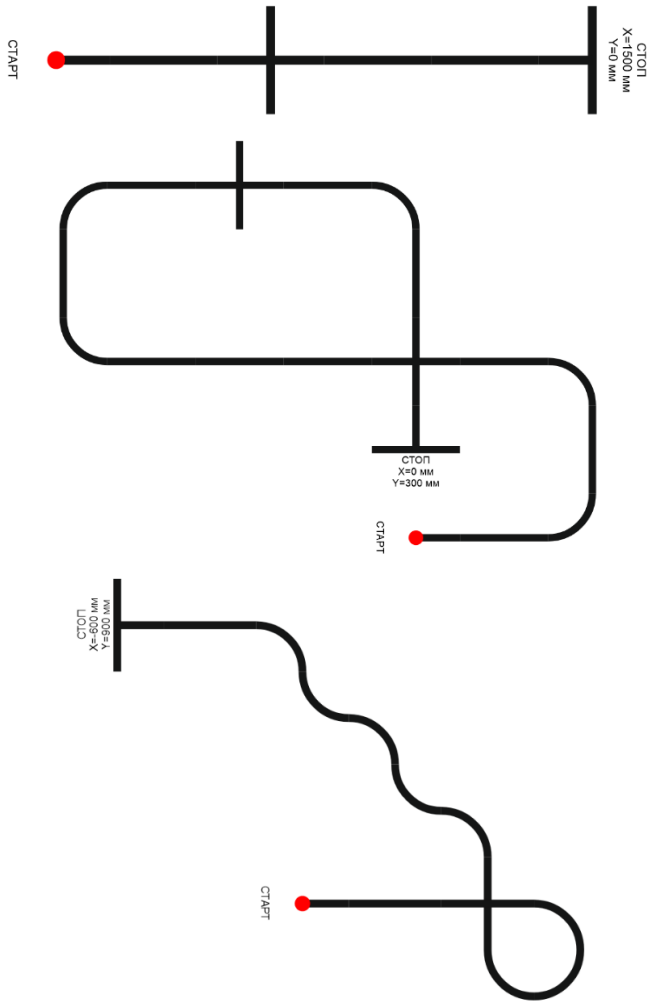
Общий размер полигона составляет 4 х 4 метра.

Стартовая секция и стартовое положение робота - точно в центре полигона.

Ширина линии не менее 25 мм.

Гарантируется, что на старте линия расположена между датчиками A2 и A3. Гарантируется, что первые 300 мм линии - прямой участок. Далее следуют повороты с радиусом не менее 150 мм. Гарантируется, что боковые линии на перекрестках отстоят от основной линии не менее чем на 125 мм и имеют ширину не менее 25 мм.

Пример полей:



Примеры заданий и тестовый проект:

В приложенном архиве находятся следующие фалы:

- Test_field_1_exercise.qrs / Test_field_2_exercise.qrs / Test_field_3_exercise.qrs - файлы-упражнения с настроенными тренировочными полями;

- input.txt - файл со входными данными для программы;
- task_sen_j.py - файл-программа с исходным кодом участника.

Файлы-упражнения могут использоваться для отладки программ. После отладки код программы необходимо перенести в файл task_sen_j.py и отправить в качестве решения задачи.

Данные:

Параметры движения содержатся в файле input.txt

В первой строке файла указано целое число от 1 до 100: номер перекрестка, на котором требуется остановиться роботу.

Решение

Задача в значительной мере похожа на задачу F возрастной группы 7-8 классов. Единственное отличие – количество перекрестков для преодоления неизвестно заранее и считывается из файла. В примере кода, предоставляемом организаторами, уже содержатся команды для считывания файла и перевода значения в число. Далее запускается цикл с движением по линии до перекрестка и гарантированным его проездом (аналогично задаче F из младшей возрастной группы).

Итоговый код решения может выглядеть следующим образом:

```
1. import sys
2. import time
3. import random
4. import math
5.
6. class Program():
7.   __interpretation_started_timestamp__ = time.time() * 1000
8.
9.   pi = 3.141592653589793
10.
11.        def execMain(self):
12.          Diam=0.056 #диаметр колес робота, в метрах
13.          L=0.077 #полуколея робота, в метрах
14.          X=0.0 #координата X робота, в метрах
15.          Y=0.0 #координата Y робота, в метрах
16.          #на старте обе координаты робота нулевые
17.          #во время движения рекомендуется обновлять эти переменные
18.
19.          #добавьте свой код ниже
20.          kf=0.9
21.          M=0
22.          kp=1.5
23.          ki=0
24.          kd=0
25.          err=0
26.          errold=0
27.          I=0
28.          n=0
29.          leftSpd = 0
30.          rightSpd= 0
31.          Nleft_old=0
32.          Nright_old=0
33.
34.          lines = script.readAll("input.txt")
35.          N=float(lines[0])
36.
```

```
        while n<N:
          s1 = brick.sensor("A1").read()
          s2 = brick.sensor("A2").read()
          s3 = brick.sensor("A3").read()
          s4 = brick.sensor("A4").read()
          s1f = s1
          s4f = s4
          while (s1+s4)<50:
            s1 = brick.sensor("A1").read()
            s2 = brick.sensor("A2").read()
            s3 = brick.sensor("A3").read()
            s4 = brick.sensor("A4").read()
            s1f = kf*s1f+(1-kf)*s1
            s4f = kf*s4f+(1-kf)*s4
            err = s2 - s3
            P=kp*err
            I=I+ki*err
            D=kd*(err-errold)
            U=P+I+D
            leftSpd = 50-U
            rightSpd= 50+U
            brick.motor("M3").setPower(rightSpd)
            brick.motor("M4").setPower(leftSpd)
            errold = err
            Nleft=brick.encoder("M4").read()
            Nright=brick.encoder("M3").read()
            dSleft = math.pi * Diam *(Nleft - Nleft_old)/360
            dSright = math.pi * Diam *(Nright - Nright_old)/360
            X = X+(dSright+dSleft)/2*math.cos( M + (dSright-dSleft)/(4*L) )
            Y = Y+(dSright+dSleft)/2*math.sin( M + (dSright-dSleft)/(4*L) )
            M=M+(dSright-dSleft)/(2*L)
            Nleft_old = Nleft
            Nright_old = Nright
            script.wait(10)
          n=n+1
          startTime = script.time()
          brick.playTone(200, 300)
          dt = 0
          while dt<800:
            s1 = brick.sensor("A1").read()
            s2 = brick.sensor("A2").read()
            s3 = brick.sensor("A3").read()
            s4 = brick.sensor("A4").read()
            s1f = kf*s1f+(1-kf)*s1
            s4f = kf*s4f+(1-kf)*s4
            err = s2 - s3
            P=kp*err
            I=I+ki*err
            D=kd*(err-errold)
            U=P+I+D
            leftSpd = 40-U
            rightSpd= 40+U
            brick.motor("M3").setPower(rightSpd)
            brick.motor("M4").setPower(leftSpd)
            errold = err
            Nleft=brick.encoder("M4").read()
            Nright=brick.encoder("M3").read()
            dSleft = math.pi * Diam *(Nleft - Nleft_old)/360
            dSright = math.pi * Diam *(Nright - Nright_old)/360
            X = X+(dSright+dSleft)/2*math.cos( M + (dSright-dSleft)/(4*L) )
            Y = Y+(dSright+dSleft)/2*math.sin( M + (dSright-dSleft)/(4*L) )
            M=M+(dSright-dSleft)/(2*L)
            Nleft_old = Nleft
            Nright_old = Nright
            script.wait(10)
            dt=script.time()-startTime

        brick.motor("M3").powerOff()
        brick.motor("M4").powerOff()

        script.wait(10)
        brick.stop()
```

```
109.          return
110.
111.      def main():
112.        program = Program()
113.        program.execMain()
114.
115.      if __name__ == '__main__':
116.        main()
```

Задача I. Одометрия и перемещение по точкам

В симуляторе TRIK Studio подготовить управляющую программу, перемещающую робота между точками по заданным их координатам. Координаты точек считываются из файла.

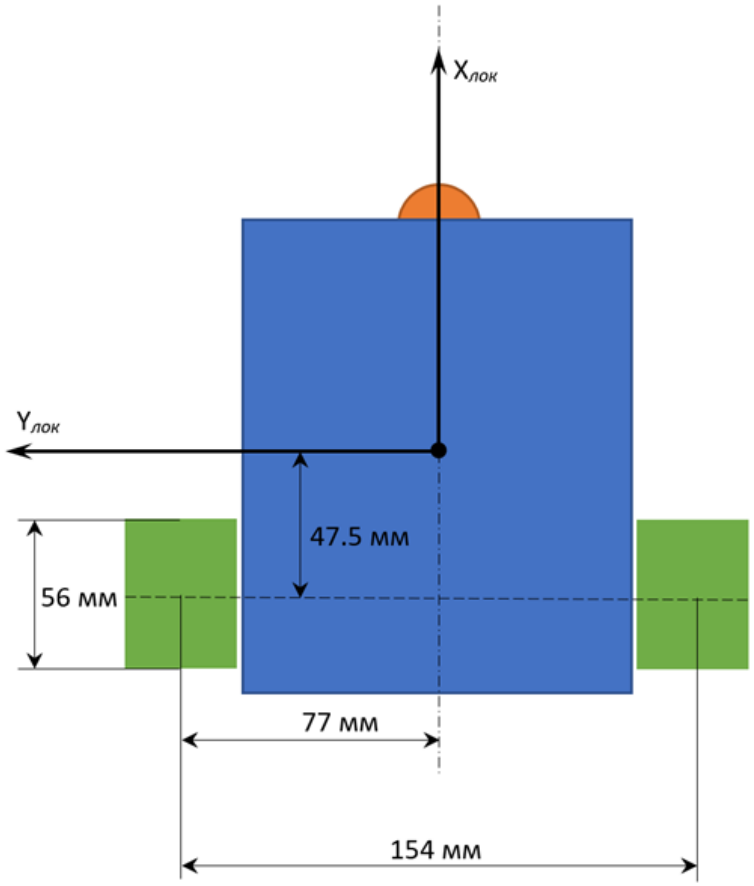
Управляющая программа проверяется на нескольких полях, имеющих разную конфигурацию точек. За каждое поле начисляется до 10% баллов. Поле засчитывается при совпадении следующих условий:

- "посещение первой точки" - робот выехал из стартовой зоны и приехал в зону диаметром 300 мм с центром в первой заданной точке;
- "посещение N-ой точки" - выполнено условие "посещение N-1 точки" и робот приехал в зону с диаметром 300 мм с центром в N-ой точке;
- "финиш в последней точке" - выполнено условие "посещение предпоследней точки" и центр робота находится в зоне диаметром 300 мм, центр которой совпадает с последней точкой;
- скорости обоих моторов робота равны нулю.

Описание робота:

Дифференциальная платформа.

Размеры модели в симуляторе TRIK Studio представлены на рисунке.



Левый мотор робота подключен к порту M4, правый мотор - к порту M3.

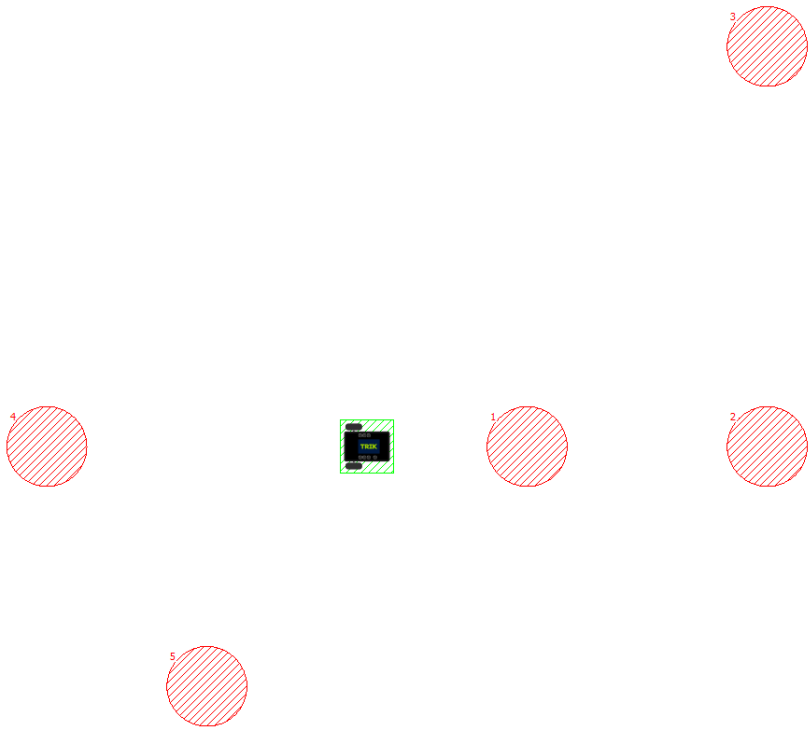
Энкодеры робота возвращают угол поворота колес в градусах. Положительные числа обозначают вращение колеса, обеспечивающего движение робота вперед, отрицательные - назад.

Полигон:

Общий размер полигона составляет 4 х 4 метра.

Стартовая секция и стартовое положение робота - точно в центре полигона, вдоль оси OX в положительном направлении (визуально отображается как направление в правую сторону). Центр полигона (стартовая точка робота) имеет координаты (0, 0).

Пример полигона:



Пример файла input.txt для примера полигона:

600 0

1500 0

1500 1500

-1200 0

-600 -900

Примеры заданий и тестовый проект:

В приложенном архиве находятся следующие фалы:

- Test_field_exercise.qrs - файл-упражнение с настроенным тренировочным полем;
- input.txt - файл со входными данными для программы;
- task_sen_k.py - файл-программа с исходным кодом участника.

Файлы-упражнения могут использоваться для отладки программ. После отладки код программы необходимо перенести в файл task_sen_k.py и отправить в качестве решения задачи.

Данные:

Параметры движения содержатся в файле input.txt

В первой строке файла указаны два целых числа от -2000 до 2000, разделенных пробелом: координаты X и Y первой точки для посещения в миллиметрах. Числа могут быть как положительными, так и отрицательными.

В следующих строках указаны аналогичные пары чисел, обозначающие координаты следующих точек для посещения. Ограничения и значения чисел аналогичны таким же из первой строки. Количество строк соответствует количеству точек для посещения.

Решение

Задача подразумевает способность робота определять свои координаты на любом этапе движения или выполнения программы. Единственные доступные в программе датчики – энкодеры моторов. Вычисление координат робота в зависимости от показаний энкодеров называется одометрией. Подробно о ней рассказано в видео «Дифференциальная платформа роботов». В текущей задаче можно воспользоваться полными формулами из видео или упрощенными, предполагая, что при малых «шагах» (перемещениях робота) длина хорды примерно равна длине дуги.

С учетом этих формул можно подготовить следующий набор функций:

1. Для вычисления разницы координат между текущим положением робота и следующей заданной точкой;
2. Для поворота робота из его текущей ориентации к следующей заданной точке;
3. Для вычисления расстояния между текущим положением робота и следующей заданной точкой;
4. Для проезда с синхронизацией моторов на заданное расстояние.

Вызывая эти функции в указанном порядке для каждой точки из файла, можно добиться решения задачи. Считывание файла и перевод текста в числа (координаты) дан в примере кода от организаторов.

Итоговая программа на ЯП Python может иметь следующий вид:

```
1. import sys
2. import time
3. import random
4. import math
5.
6. class Program():
7.   __interpretation_started_timestamp__ = time.time() * 1000
8.
9.   pi = 3.141592653589793
10.        Diam=0.056 #диаметр колес робота, в метрах
```

```
        L=0.077 #полуколея робота, в метрах
        X=0.0 #координата X робота, в метрах
        Y=0.0 #координата Y робота, в метрах
        orient = 0.0 #ориентация робота, в градусах или радианах на усмотрение участника
        #на старте все координаты робота нулевые
        #нулевая ориентация означает направление вдоль оси OX
        #во время движения рекомендуется обновлять эти переменные

        encL=0
        encR=0
        Nleft_old=0
        Nright_old=0

        def execMain(self):
          #добавьте свой код ниже
          self.X=-0.0475
          points = script.readAll("solutions/input.txt")
          for point in points:
            pnt = [int(i) for i in point.split()]
            pointX = pnt[0]/1000
            pointY = pnt[1]/1000
            print(pointX, pointY)
            self.RotateToPoint(pointX, pointY)
            dist = self.GetDistFromPoints(self.X, self.Y, pointX, pointY)
            self.SyncMoveDist(dist)


          brick.stop()
          return

        def RotateToPoint(self, pointX, pointY):
          deltaX = pointX - self.X
          deltaY = pointY - self.Y
          angleRad = math.atan2(deltaY, deltaX)-self.orient
          while angleRad<0:
            angleRad=2*math.pi+angleRad
          angleDeg = angleRad*180/math.pi
          startEncL = brick.encoder("M4").read()
          startEncR = brick.encoder("M3").read()
          pathAngle = 0
          betta = (360*angleRad*self.L)/(math.pi*self.Diam)
          baseSPD=15
          while pathAngle < math.fabs(betta):
            pathL=brick.encoder("M4").read()-startEncL
            pathR=brick.encoder("M3").read()-startEncR
            pathAngle = ( math.fabs(pathL) + math.fabs(pathR) )/2
            err = pathR + pathL
            P=1.0*err
            U=P
            brick.motor("M4").setPower(-baseSPD-U)
            brick.motor("M3").setPower(baseSPD-U)
            Nleft=brick.encoder("M4").read()
            Nright=brick.encoder("M3").read()
            dSleft = math.pi * self.Diam *(Nleft - self.Nleft_old)/360
            dSright = math.pi * self.Diam *(Nright - self.Nright_old)/360
            self.X = self.X+(dSright+dSleft)/2*math.cos( self.orient + (dSright-dSleft)/(4*self.L) )
            self.Y = self.Y+(dSright+dSleft)/2*math.sin( self.orient + (dSright-dSleft)/(4*self.L) )
            self.orient = self.orient+(dSright-dSleft)/(2*self.L)
            self.Nleft_old = Nleft
            self.Nright_old = Nright
          brick.motor("M4").powerOff()
          brick.motor("M3").powerOff()

        def SyncMoveDist(self, dist):
          startEncL = brick.encoder("M4").read()
          startEncR = brick.encoder("M3").read()
          pathAngle = 0
          alpha = (360*dist)/(math.pi*self.Diam)
          baseSPD=35
```

```
80.           while pathAngle < math.fabs(alpha):
81.             pathL=brick.encoder("M4").read()-startEncL
82.             pathR=brick.encoder("M3").read()-startEncR
83.             pathAngle = (pathL+pathR)/2
84.             err = pathL-pathR
85.             P=1.0*err
86.             U=P
87.             brick.motor("M4").setPower(baseSPD-U)
88.             brick.motor("M3").setPower(baseSPD+U)
89.             Nleft=brick.encoder("M4").read()
90.             Nright=brick.encoder("M3").read()
91.             dSleft = math.pi * self.Diam *(Nleft - self.Nleft_old)/360
92.             dSright = math.pi * self.Diam *(Nright - self.Nright_old)/360
93.             self.X = self.X+(dSright+dSleft)/2*math.cos( self.orient +
    (dSright-dSleft)/(4*self.L) )
94.             self.Y = self.Y+(dSright+dSleft)/2*math.sin( self.orient +
    (dSright-dSleft)/(4*self.L) )
95.             self.orient = self.orient+(dSright-dSleft)/(2*self.L)
96.             self.Nleft_old = Nleft
97.             self.Nright_old = Nright
98.           brick.motor("M4").powerOff()
99.           brick.motor("M3").powerOff()
100.
101.        def SyncRotateToAngle(self, angle):
102.          pass
103.
104.        def GetDistFromPoints(self, firstPointX, firstPointY, secondPointX,
    secondPointY):
105.          dist = math.sqrt( (secondPointX-firstPointX)**2 + (secondPointY-
    firstPointY)**2 )
106.          return dist
107.
108.      def main():
109.        program = Program()
110.        program.execMain()
111.
112.      if __name__ == '__main__':
113.        main()
```

Задача J. Перекрестки

В симуляторе TRIK Studio подготовить управляющую программу, перемещающую робота по линиям, образующим сетку, в заданные перекрестки.

Управляющая программа проверяется на одном поле, но с разными последовательностями посещения перекрестков. За каждое поле начисляется до 10% баллов. При проверке на каждом поле начисляется по 2 балла за каждый посещенный перекресток при совпадении следующих условий:

- робот переместился в перекресток, указанный ему для посещения, то есть любая точка проекции робота находится над центром указанного перекрестка;
- робот переместился в указанный перекресток в верном порядке, то есть предыдущий указанный перекресток был засчитан, либо это первый указанный по порядку посещения перекресток;
- скорости обоих моторов робота равны нулю.

Описание робота:

Дифференциальная платформа. Робот оснащен четырьмя датчиками отраженного света, направленными вниз:

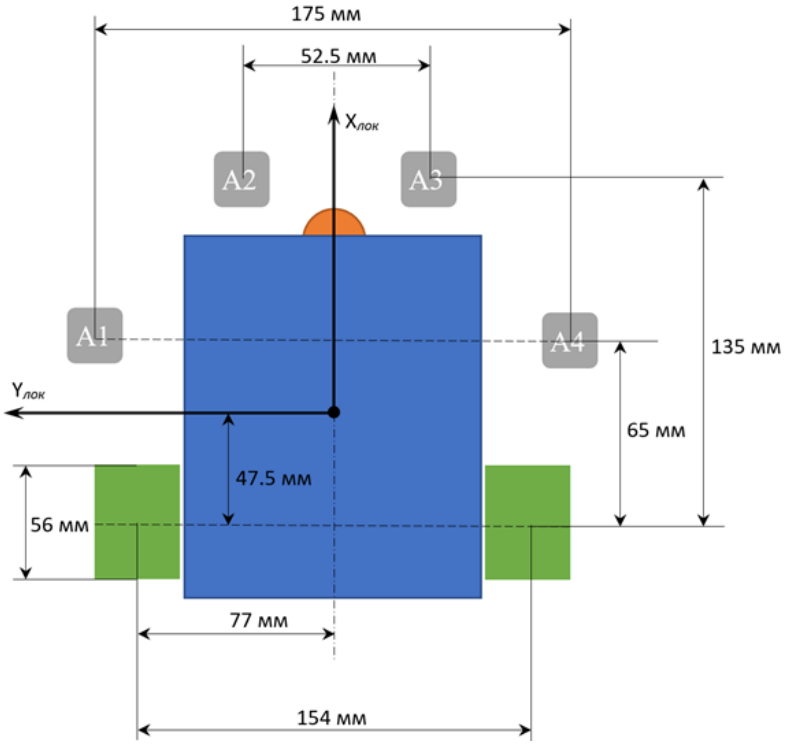
К порту A1 подключен левый боковой датчик.

К порту A2 подключен левый передний датчик.

К порту A3 подключен правый передний датчик.

К порту A4 подключен правый боковой датчик.

Размеры модели в симуляторе TRIK Studio представлены на рисунке.



Левый мотор робота подключен к порту M4, правый мотор - к порту M3.

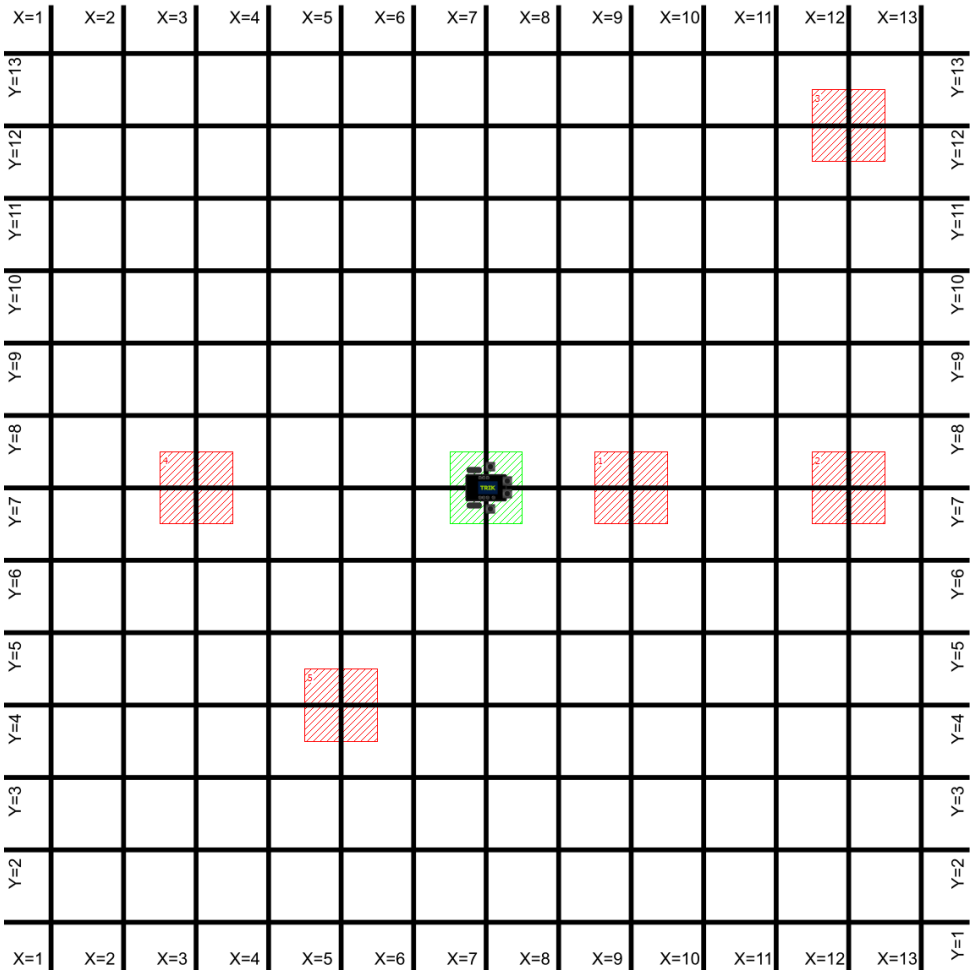
Энкодеры робота возвращают угол поворота колес в градусах. Положительные числа обозначают вращение колеса, обеспечивающего движение робота вперед, отрицательные - назад.

Полигон:

Общий размер полигона составляет 4 x 4 метра. На полигоне нанесены прямые линии, образующие на пересечениях перекрестки. Толщина линий - 20 мм, расстояние между центрами линий - 300 мм. Перекрестки имеют координаты по осям X и Y - их порядковые номера. Всего полигон разделен на 13 перекрестков по оси X и на 13 перекрестков по оси Y.

Робот начинает движение в центральном перекрестке с координатами (7, 7) - ровно в середине полигона. Робот направлен вдоль оси OX в положительном направлении (визуально отображается как направление в правую сторону).

Пример полигона:



Пример файла input.txt для примера полигона:

9 7

12 7

12 12

3 7

5 4

Примеры заданий и тестовый проект:

В приложенном архиве находятся следующие фалы:

- Test_field_exercise.qrs - файл-упражнение с настроенным тренировочным полем;
- input.txt - файл со входными данными для программы;
- task_sen_l.py - файл-программа с исходным кодом участника.

Файлы-упражнения могут использоваться для отладки программ. После отладки код программы необходимо перенести в файл task_sen_l.py и отправить в качестве решения задачи.

Данные:

Параметры движения содержатся в файле input.txt

В первой строке файла указаны два целых числа от 1 до 13, разделенных пробелом: координаты X и Y первого перекрестка для посещения. В следующих строках указаны аналогичные пары чисел, обозначающие координаты следующих перекрестков для посещения. Ограничения и значения чисел аналогичны таким же из первой строки. Количество строк соответствует количеству перекрестков для посещения.

Решение

См. решение задачи J возрастной группы 7-8 классов.