

Второй отборочный этап Старшая возрастная группа

Задача G. Движение по линии

Задание

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

Описание робота

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

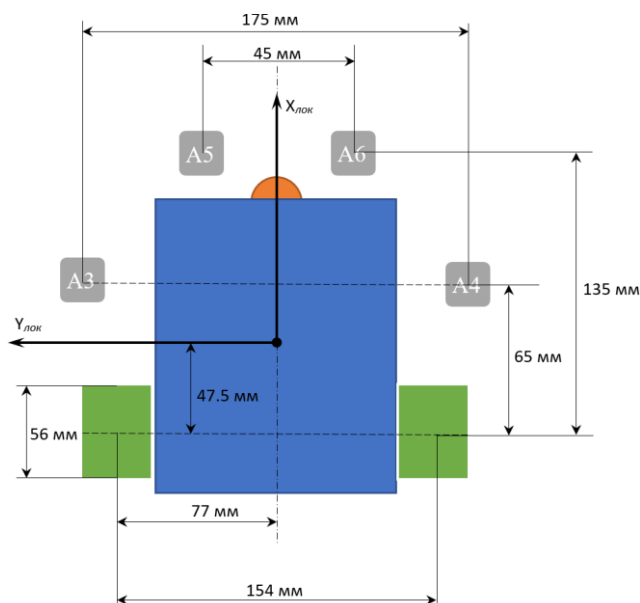
К порту A3 подключен левый боковой датчик.

К порту A5 подключен левый передний датчик.

К порту A6 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.

Реалистичные физика и моторы включены. Реалистичные сенсоры отключены.

Полигон

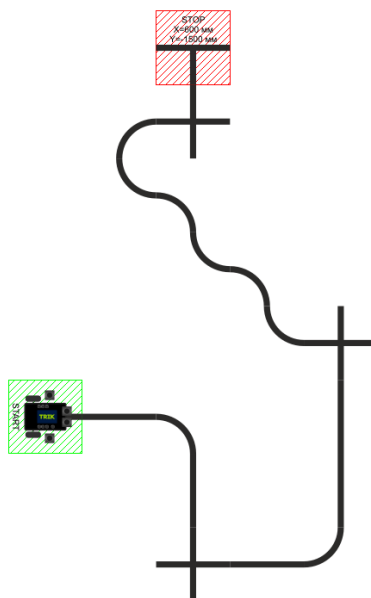
Общий размер полигона составляет 4 x 4 метра.

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 A5 и A6. 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:



Начисление баллов

Управляющая программа проверяется на пяти полигонах: одном известном заранее (из упражнения) и четырех неизвестных, имеющих разную конфигурацию поворотов, прямых участков и перекрестков. За каждое поле начисляется до 20% баллов. Полигон имеет следующие события, происходящие при совпадении следующих условий:

- "Out from start" - выводит сообщение в консоль "Out from start"; требуемые условия:
 - ни одна точка проекции робота не находится над зоной старта;
 - скорости левого и правого моторов ненулевые.
- "Finish checker" - начисляет 10 баллов и выводит сообщение в консоль "Successful finish"; завершает проверку; требуемые условия:
 - ранее наступило событие "Out from start";
 - центр робота находится в финишной зоне;
 - скорости обоих моторов робота равны нулю.

Примеры заданий и тестовый проект:

В приложенном архиве находятся следующие файлы:

- sen_task_3_exercise.qrs - файл-упражнение с настроенным тренировочным полем;
- sen_task_3.py - файл-программа с исходным кодом участника.

Файлы-упражнения могут использоваться для отладки программ. После отладки код программы необходимо перенести в файл sen_task_3.py и отправить в качестве решения задачи.

Решение:

За основу решения взято решение задач F и J младшей возрастной группы второго отборочного этапа олимпиады сезона 2024-25 гг (их можно найти в материалах подготовки на официальном сайте олимпиады). Изменены порты подключения датчиков, незначительно изменены коэффициенты и пороговые значения для определения перекрёстков и линий при поворотах.

Итоговый код программы:

```
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 #полуколя робота, в метрах
12.    kf=0.9
13.    kp=1.0
14.    ki=0
15.    kd=0
16.    errold=0
17.    I=0
18.    s1f=0
19.    s2f=0
20.    s3f=0
21.    s4f=0
22.
23.    def execMain(self):
24.        #добавьте свой код ниже
25.        self.Crossroads(4)
26.
27.        #остановка моторов на финише
28.        brick.motor("M3").powerOff()
29.        brick.motor("M4").powerOff()
30.
31.        brick.stop()
32.        return
33.
34.    def RotateToLine(self):
35.        brick.motor("M3").setPower(25)
36.        brick.motor("M4").setPower(-25)
37.        script.wait(1000)
38.        s5 = brick.sensor("A5").read()
39.        while s5<30:
40.            s5 = brick.sensor("A5").read()
41.        brick.motor("M4").powerOff()
42.        brick.motor("M3").powerOff()
43.
44.    def LineTrace(self):
45.        s5 = brick.sensor("A5").read()
46.        s6 = brick.sensor("A6").read()
47.        err = s5 - s6
48.        P=self.kp*err
49.        self.I=self.I+self.ki*err
50.        D=self.kd*(err-self.errold)
51.        U=P+self.I+D
52.        leftSpd = 50-U
53.        rightSpd= 50+U
54.        brick.motor("M3").setPower(rightSpd)
55.        brick.motor("M4").setPower(leftSpd)
56.        self.errold = err
57.        script.wait(10)
58.        pass
```

```
59.
60.     def LineToCrossroad(self):
61.         s3 = brick.sensor("A3").read()
62.         s4 = brick.sensor("A4").read()
63.         while (s3+s4)<60:
64.             s3 = brick.sensor("A3").read()
65.             s4 = brick.sensor("A4").read()
66.             self.LineTrace()
67.             brick.motor("M4").powerOff()
68.             brick.motor("M3").powerOff()
69.
70.     def LineToDist(self, dist):
71.         startEncL = brick.encoder("M4").read()
72.         startEncR = brick.encoder("M3").read()
73.         pathAngle=0
74.         alpha = (360*dist)/(math.pi*self.Diam)
75.         while pathAngle < math.fabs(alpha):
76.             pathL=brick.encoder("M4").read()-startEncL
77.             pathR=brick.encoder("M3").read()-startEncR
78.             pathAngle = (pathL+pathR)/2
79.             self.LineTrace()
80.             brick.motor("M4").powerOff()
81.             brick.motor("M3").powerOff()
82.
83.     def Crossroads(self, crossNum):
84.         n=0
85.         while n<crossNum:
86.             self.LineToCrossroad()
87.             self.LineToDist(0.0475)
88.             n=n+1
89.             self.RotateToLine()
90.             brick.playTone(1000, 100)
91.
92.     def main():
93.         program = Program()
94.         program.execMain()
95.
96. if __name__ == '__main__':
97.     main()
```

Задача Н. Стена

Задание

В симуляторе TRIK Studio подготовить управляющую программу, способную перемещать робота вдоль линии и вдоль стены, переключаясь между этими режимами. Режимы работы идут в следующей последовательности: сперва движение вдоль линии, потом вдоль стены, и в конце опять вдоль линии до перекрестка.

Описание робота

Дифференциальная платформа. Робот оснащен пятью датчиками:

К порту A5 подключен левый датчик линии.

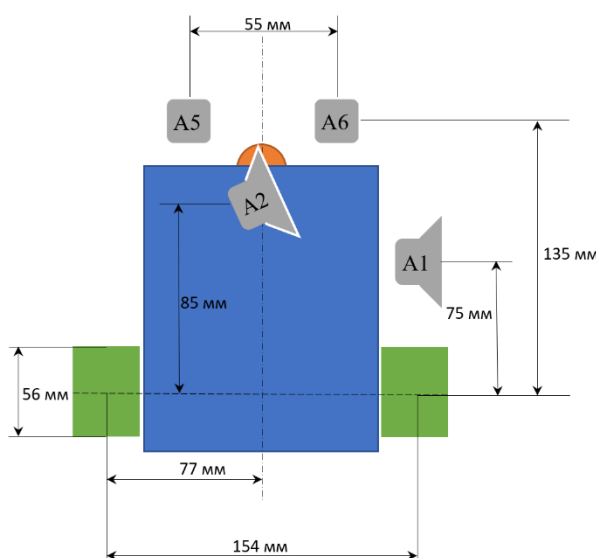
К порту A6 подключен правый датчик линии.

К порту A4 подключен центральный датчик линии.

К порту A1 подключен датчик расстояния, направленный направо под 90 градусов относительно направления движения робота.

К порту A2 подключен датчик расстояния, направленный направо под 75 градусов относительно направления движения робота.

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.

Реалистичные физика и моторы включены. Реалистичные сенсоры отключены.

Полигон

Общий размер полигона составляет 4 x 4 метра.

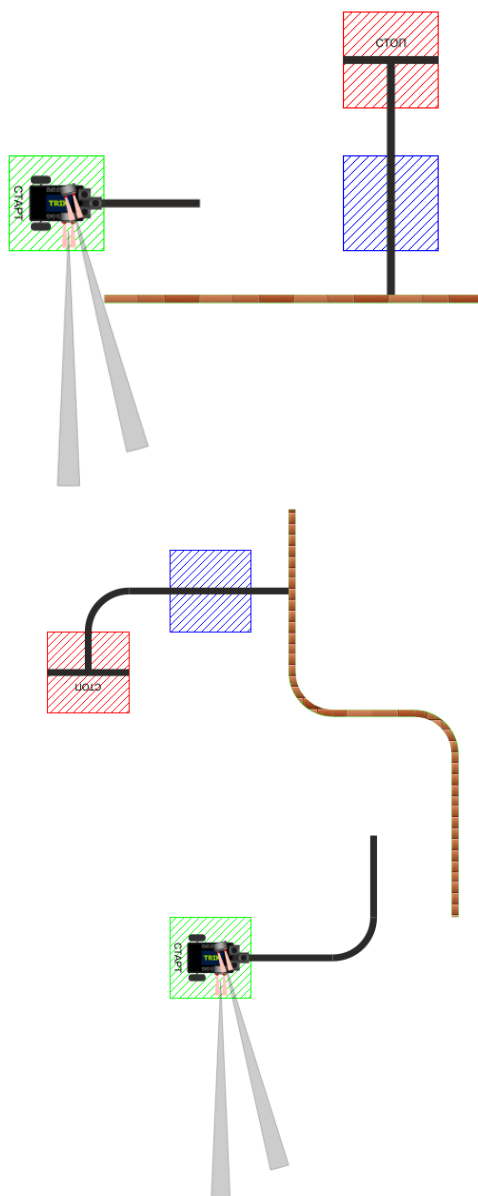
Стартовая секция и стартовое положение робота - точно в центре полигона, в направлении вправо.

Ширина линии не менее 25 мм.

Гарантируется, что на старте линия расположена между датчиками A5 и A6. 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. Гарантируется, что при движении по линии со старта стена появится справа от робота на расстоянии 300 мм; к концу первого участка линии справа будет участок стены длиной не менее 300 мм. Гарантируется, что за 300 мм до завершения стены будет поперечная линия, на которой роботу потребуется повернуть налево.

Примеры полей:



Зеленым обозначена зона старта - ее центр совпадает с центром робота на старте; размеры зоны 300x300 мм.

Синим обозначена промежуточная зона - ее центр совпадает с линией завершения движения вдоль стены, находится на расстоянии 300 мм от стены; размеры зоны 300x300 мм.

Красным обозначена зона финиша - ее центр совпадает с центром перекрестка; размеры зоны 300x300 мм.

Начисление баллов

Управляющая программа проверяется на пяти полигонах: двух известных заранее (из упражнений) и трех неизвестных, имеющих разную конфигурацию поворотов, прямых участков

и стен. За каждое поле начисляется до 20% баллов. Полигон имеет следующие события, происходящие при совпадении следующих условий:

- "Out from start" - выводит сообщение в консоль "Out from start"; требуемые условия:
 - ни одна точка проекции робота не находится над зоной старта;
 - скорости левого и правого моторов ненулевые.
- "Intermediate zone checker"- начисляет 10 баллов и выводит сообщение "Successful movement along the wall"; требуемые условия:
 - ранее наступило событие "Out from start";
 - центр проекции робота находится в промежуточной зоне.
- "Finish checker" - начисляет 5 баллов и выводит сообщение в консоль "Successful finish"; завершает проверку; требуемые условия:
 - центр робота находится в финишной зоне;
 - скорости обоих моторов робота равны нулю.

Примеры заданий и тестовый проект

В приложенном архиве находятся следующие файлы:

- `sen_task_4_1_exercise.qrs` / `sen_task_4_2_exercise.qrs` - файлы-упражнения с настроенными тренировочными полями;
- `sen_task_4.py` - файл-программа с исходным кодом участника.

Файлы-упражнения могут использоваться для отладки программ. После отладки код программы необходимо перенести в файл `sen_task_4.py` и отправить в качестве решения задачи.

Решение

Задача подразумевает движение по линии и вдоль стены, а также переключение между этими режимами в зависимости от внешних условий.

Для удобства написания кода в начале программы можно определить обращения к моторам и датчикам с помощью более коротких и понятных имён (см. строки 6-14 в итоговом коде программы).

Движение по линии рассмотрено в предыдущей задаче. Его удобнее вынести в отдельную функцию `LineTrace()` и вызывать ее по необходимости в других функциях, например, с прописанными условиями переключения.

Движение вдоль стены можно реализовать аналогичным образом, с помощью регулятора. Причем модель робота содержит два датчика расстояния, участники могут выбрать наиболее подходящий. В дальнейшем решение будет строиться на использовании датчика на порту A2, так как он направлен немного вперёд (это облегчает движение вдоль стены). Ошибка («невязка») регулятора будет вычисляться как разница между показаниями датчика и заданным значением – желаемым расстоянием до стены. Так как линия находится на расстоянии 30 см от стены, то и за желаемое расстояние примем значение 30. Движение вдоль стены удобнее вынести в отдельную функцию `WallTrace()`.

Переключение режимов можно производить по следующим правилам:

- Робот начинает движение по линии, и выполняет его до тех пор, пока линия не закончится, то есть пока все три датчика суммарно видят более какого-то порогового значения (если все три датчика видят белое, то показания на них упадут до околонулевых значений). Реализовано функцией *LineToEnd()*.
- После окончания линии робот переключается на движение вдоль стены и выполняет его до тех пор, пока центральный датчик (расположенный впереди всех остальных) не увидит черную линию. Реализовано функцией *WallToLine()*.
- После обнаружения линии центральным датчиком робот выполняет проезд вперед с синхронизацией моторов на 15 см (0.15 м), чтобы оказаться колесами примерно на линии (в этом состоянии удобнее делать поворот с поиском линии). Реализовано функцией *SyncToDist()*.
- Робот совершает поворот направо до тех пор, пока не увидит правым датчиком черную линию, то есть пока не окажется датчиками по разные стороны от линии. Реализовано функцией *RotateToLine()*.
- Робот совершает проезд по линии до перекрёстка. Реализовано функцией *LineToCrossroad()*.

Итоговый код программы может иметь следующий вид:

```

1. import sys
2. import time
3. import random
4. import math
5.
6. centrLine = brick.sensor("A4").read
7. leftLine = brick.sensor("A5").read
8. rightLine = brick.sensor("A6").read
9. distSens1 = brick.sensor("A1").read
10. distSens2 = brick.sensor("A2").read
11. leftMotor = brick.motor("M4")
12. rightMotor = brick.motor("M3")
13. leftEnc = brick.encoder("M4").read
14. rightEnc = brick.encoder("M3").read
15.
16. class Program():
17.     __interpretation_started_timestamp__ = time.time() * 1000
18.
19.     pi = 3.141592653589793
20.     Diam=0.056 #diameter of the robot's wheels, in meters
21.     L=0.077 #half-track of the robot, in meters
22.     drivespd=50
23.     turnspd=25
24.     kf=0.9
25.     kp_line=0.9
26.     ki_line=0
27.     kd_line=0
28.     kp_wall=1.9
29.     ki_wall=0
30.     kd_wall=0.1
31.     kp_enc=1.0
32.     ki_enc=0
33.     kd_enc=0
34.     errold_line=0
35.     errold_wall=0
36.     errold_enc=0
37.     I_line=0
38.     I_wall=0
39.     I_enc=0
40.

```

```

41. def execMain(self):
42.     #add your code here
43.     self.LineToEnd()
44.     print("Конец линии, переключение на стену")
45.     self.wallToLine()
46.     print("Конец стены, поворот на линию")
47.     self.SyncToDist(0.15)
48.     self.RotateToLine()
49.     print("Движение по линии")
50.     self.LineToCrossroad()
51.
52.     #motor stop on finish
53.     brick.motor("M3").poweroff()
54.     brick.motor("M4").poweroff()
55.     script.wait(10)
56.
57.     brick.stop()
58.     return
59.
60. def RotateToLine(self):
61.     rightMotor.setPower(self.turnSpd)
62.     leftMotor.setPower(-self.turnSpd)
63.     script.wait(1000)
64.     while rightLine()<30:
65.         pass
66.     leftMotor.poweroff()
67.     rightMotor.powerOff()
68.
69. def LineTrace(self):
70.     err = leftLine() - rightLine()
71.     P=self.kp_line*err
72.     self.I_line=self.I_line+self.ki_line*err
73.     D=self.kd_line*(err-self.erold_line)
74.     U=P+self.I_line+D
75.     leftMotor.setPower(self.drivespd-U)
76.     rightMotor.setPower(self.drivespd+U)
77.     self.erold_line = err
78.     script.wait(10)
79.     pass
80.
81. def LineToEnd(self):
82.     while leftLine()+centrLine()+rightLine()>5:
83.         self.LineTrace()
84.         leftMotor.poweroff()
85.         rightMotor.powerOff()
86.         erold_line=0
87.         I_line=0
88.
89. def wallToLine(self):
90.     while centrLine() <50:
91.         self.wallTrace()
92.         leftMotor.powerOff()
93.         rightMotor.powerOff()
94.
95. def wallTrace(self):
96.     err = distSens2() - 30
97.     P=self.kp_wall*err
98.     self.I_wall=self.I_wall+self.ki_wall*err
99.     D=self.kd_wall*(err-self.erold_wall)
100.    U=P+self.I_wall+D
101.    rightMotor.setPower(self.drivespd - U)
102.    leftMotor.setPower(self.drivespd + U)
103.    self.erold_wall = err
104.    script.wait(10)
105.
106. def LineToCrossroad(self):

```

```

107.         while ( leftLine() + rightLine() )<60:
108.             self.LineTrace()
109.             leftMotor.powerOff()
110.             rightMotor.powerOff()
111.
112.         def syncToDist(self, dist):
113.             brick.encoder("E3").reset()
114.             brick.encoder("E4").reset()
115.             pathAngle=0
116.             alpha = (360*dist)/(math.pi*self.Diam)
117.             while pathAngle < math.fabs(alpha):
118.                 pathL= leftEnc()
119.                 pathR=rightEnc()
120.                 pathAngle = (pathL+pathR)/2
121.                 self.SyncMove()
122.             leftMotor.powerOff()
123.             rightMotor.powerOff()
124.
125.         def syncMove(self):
126.             err = leftEnc() - rightEnc()
127.             P=self.kp_enc*err
128.             self.I_enc=self.I_enc+self.ki_enc*err
129.             D=self.kd_enc*(err-self.errold_enc)
130.             U=P+self.I_enc+D
131.             rightMotor.setPower(self.drivespd + U)
132.             leftMotor.setPower(self.drivespd - U)
133.             self.errold_enc = err
134.             script.wait(10)
135.
136.         def LineToDist(self, dist):
137.             startEncL = leftEnc()
138.             startEncR = rightEnc()
139.             pathAngle=0
140.             alpha = (360*dist)/(math.pi*self.Diam)
141.             while pathAngle < math.fabs(alpha):
142.                 pathL=leftEnc()-startEncL
143.                 pathR=rightEnc()-startEncR
144.                 pathAngle = (pathL+pathR)/2
145.                 self.LineTrace()
146.             leftMotor.powerOff()
147.             rightMotor.powerOff()
148.
149.         def main():
150.             program = Program()
151.             program.execMain()
152.
153.         if __name__ == '__main__':
154.             main()

```

Задача I. Движение по границе цветов

Задание

В симуляторе TRIK Studio подготовить управляющую программу, перемещающую робота вдоль границы цветных зон. Зоны могут быть красного и зеленого цвета, но на старте неизвестно, какая зона находится слева или справа от робота.

Описание робота

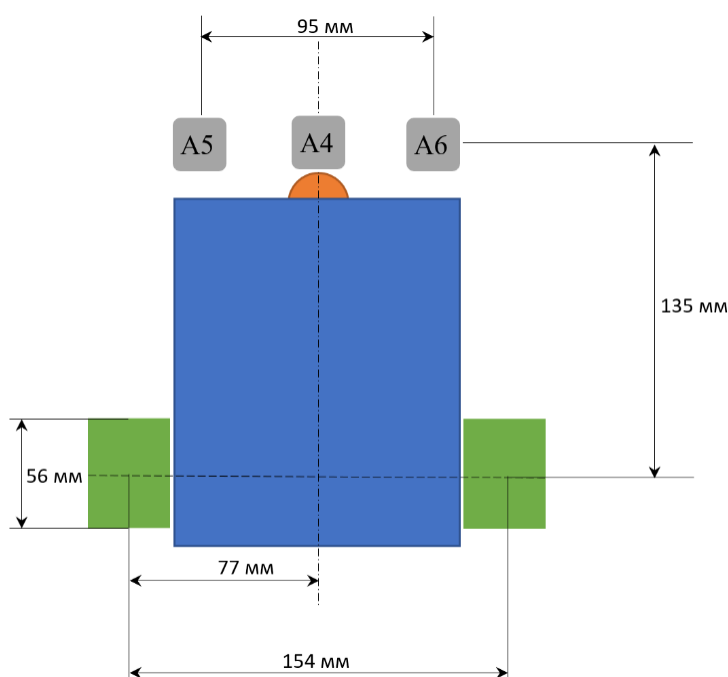
Дифференциальная платформа. Робот оснащен тремя датчиками отраженного света, направленными вниз:

К порту A5 подключен левый датчик.

К порту A6 подключен правый датчик.

К порту A4 подключен средний 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.

Реалистичные физика и моторы включены. Реалистичные сенсоры отключены.

Полигон

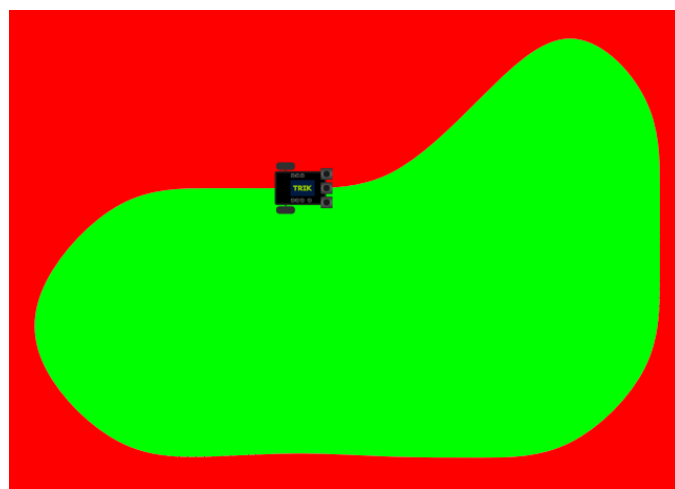
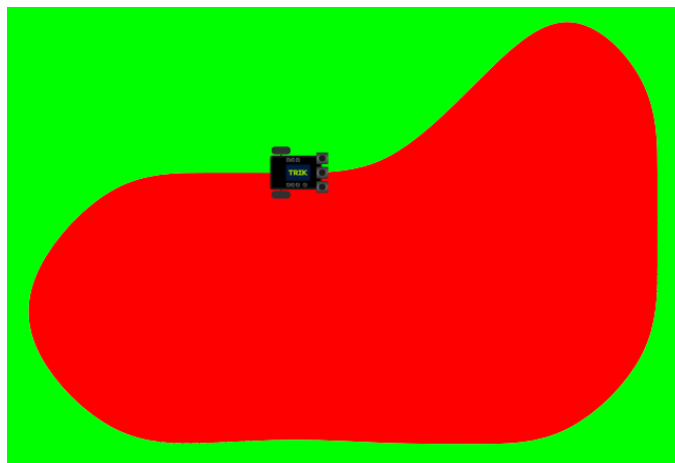
Общий размер полигона составляет 4 x 4 метра.

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

На полигоне присутствуют зоны двух цветов: красного (код ff0000) и зелёного (код 00ff00). Гарантируется, что граница цветов образует замкнутую линию, при движении по которой робот может вернуться к зоне старта.

Гарантируется, что на старте левый датчик робота установлен над зоной одного цвета, правый - над зоной другого цвета, а средний - над границами цветов. Гарантируется, что первые 150 мм границы линий - прямой участок.

Примеры полей:



Зона старта - зона размером 300x300 мм, центр которой совпадает с центром полигона и робота в момент старта.

Промежуточные зоны 1 и 2 - зоны размером 300x300 мм, центры которых совпадают с границей цветовых зон (промежуточные зоны расположены на прямых участках границы цветных зон).

Начисление баллов

Управляющая программа проверяется на пяти полигонах: одном известном заранее (из упражнений) и четырех неизвестных, имеющих разную конфигурацию цветовых зон. За каждое поле начисляется до 20% баллов. Полигон имеет следующие события, происходящие при совпадении следующих условий:

- "Out from start" - выводит сообщение в консоль "Out from start"; требуемые условия:
 - ни одна точка проекции робота не находится над зоной старта;
 - скорости левого и правого моторов ненулевые.
- "Intermediate 1 zone checker" - начисляет 5 баллов и выводит сообщение "The robot arrived at the first intermediate zone"; требуемые условия:
 - ранее наступило событие "Out from start";
 - центр проекции робота находится в промежуточной зоне 1.

- "Intermediate 2 zone checker" - начисляет 5 баллов и выводит сообщение "The robot arrived at the second intermediate zone"; требуемые условия:
 - ранее наступило событие "Intermediate 1 zone checker";
 - центр проекции робота находится в промежуточной зоне 2.
- "Finish checker" - начисляет 5 баллов и выводит сообщение в консоль "Successful finish"; завершает проверку; требуемые условия:
 - ранее наступило событие "Intermediate 2 zone checker";
 - центр робота находится в зоне старта.

Примеры заданий и тестовый проект

В приложенном архиве находятся следующие файлы:

- `sen_task_12_exercise.qrs` / `sen_task_12_exercise_alt.qrs` - файл-упражнение с настроенным тренировочным полем;
- `sen_task_12.py` - файл-программа с исходным кодом участника.

Файлы-упражнения могут использоваться для отладки программ. После отладки код программы необходимо перенести в файл `sen_task_12.py` и отправить в качестве решения задачи.

Решение

Ранее в задачах требовалось перемещать робота по черной линии на белом фоне, то есть по границе двух контрастных цветов. В этой задаче суть аналогичная – надо перемещать робота по границе контрастных цветов.

Как и в предыдущей задаче для удобства обращения к моторам и датчикам их можно определить под удобными именами в самом начале программы (см. строки 6-12 в итоговом коде программы).

Для движения будем использовать один датчик, подключенный к порту A4. На красном фоне датчики показывают примерно 25% яркости, на зелёном – примерно 75%. Среднее значение, которое датчик должен видеть на границе цветов примерно 50%. Если показания центрального датчика будут расти, значит робот уезжает на зелёный цвет и ему надо подруливать в сторону красного, и наоборот.

По условию задачи неизвестно, с какой стороны от робота находится красный и зелёный цвета. Для определения этого на старте можно использовать левый и правый датчики. Алгоритм будет следующим:

- На старте запоминаем показания левого и правого датчиков в отдельные переменные *left_light* и *right_light*.
- Движение по линии реализовано в цикле. На каждом его шаге вычисляем «левую» и «правую» ошибки как разницу между показаниями центрального датчика и запомненными в начале программы значениями *left_light* и *right_light*. Общую ошибку вычисляем как разницу абсолютных значений «левой» и «правой» ошибок. Смысл в том, чтобы «левая» и «правая» ошибки были равными по модулю, то есть значения со среднего датчика «равноудалено» от яркости цветов слева и справа от робота; а если это не так, то возникает ошибка («невязка») регулятора и он подруливает робота в нужную сторону.

- Цикл движения по линии можно сделать бесконечным, так как по условию задачи не требуется останавливаться после возврата в зону старта.

Итоговый код программы может иметь следующий вид:

```
1. import sys
2. import time
3. import random
4. import math
5.
6. centrLine = brick.sensor("A4").read
7. leftLine = brick.sensor("A5").read
8. rightLine = brick.sensor("A6").read
9. leftMotor = brick.motor("M4")
10. rightMotor = brick.motor("M3")
11. leftEnc = brick.encoder("M4").read
12. rightEnc = brick.encoder("M3").read
13.
14. class Program():
15.     __interpretation_started_timestamp__ = time.time() * 1000
16.
17.     pi = 3.141592653589793
18.     Diam=0.056 #диаметр колес робота, в метрах
19.     L=0.077 #полуколея робота, в метрах
20.     drivespd=50
21.     turnSpd=25
22.     kf=0.9
23.     kp_line=0.9
24.     ki_line=0
25.     kd_line=0
26.     errold_line=0
27.     I_line=0
28.
29.     left_light = 0
30.     right_light = 0
31.
32.     def execMain(self):
33.         #добавьте свой код ниже
34.         self.left_light = leftLine()
35.         self.right_light = rightLine()
36.         while True:
37.             self.LineTrace()
38.             #motor stop on finish
39.             brick.motor("M3").powerOff()
40.             brick.motor("M4").powerOff()
41.             script.wait(10)
42.
43.             brick.stop()
44.             return
45.
46.     def LineTrace(self):
47.         errL = centrLine() - self.left_light
48.         errR = self.right_light - centrLine()
49.         err = abs(errL) - abs(errR)
50.         P=self.kp_line*err
51.         self.I_line=self.I_line+self.ki_line*err
52.         D=self.kd_line*(err-self.errold_line)
53.         U=P+self.I_line+D
54.         leftMotor.setPower(self.drivespd-U)
55.         rightMotor.setPower(self.drivespd+U)
56.         self.errold_line = err
57.         script.wait(10)
58.         pass
59.
60.     def main():
61.         program = Program()
```

```
62.     program.execMain()
63.
64.     if __name__ == '__main__':
65.         main()
```

Задача J. Одометрия по границе цветов

Задание

В симуляторе TRIK Studio подготовить управляющую программу, перемещающую робота вдоль границы цветных зон и останавливающую после возвращения в стартовую зону. Зоны могут быть красного и зеленого цвета, но на старте неизвестно какая зона находится слева или справа от робота.

Описание робота

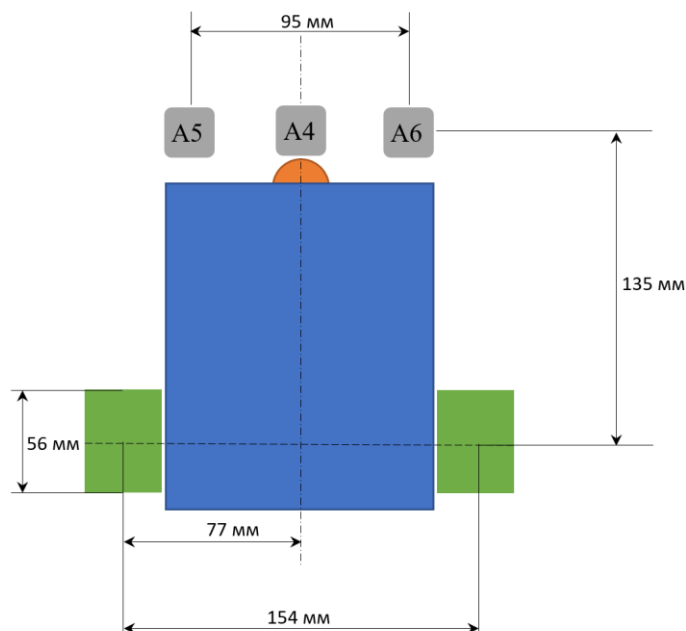
Дифференциальная платформа. Робот оснащен тремя датчиками отраженного света, направленными вниз:

К порту A5 подключен левый датчик.

К порту A6 подключен правый датчик.

К порту A4 подключен средний 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.

Реалистичные физика и моторы включены. Реалистичные сенсоры отключены.

Полигон

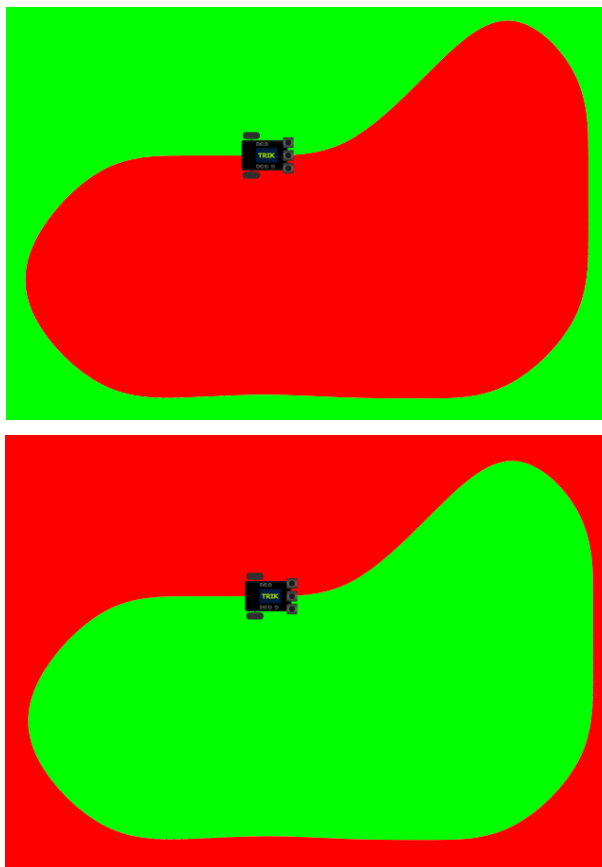
Общий размер полигона составляет 4 x 4 метра.

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

На полигоне присутствуют зоны двух цветов: красного (код ff0000) и зелёного (код 00ff00). Гарантируется, что граница цветов образует замкнутую линию, при движении по которой робот может вернуться к зоне старта.

Гарантируется, что на старте левый датчик робота установлен над зоной одного цвета, правый - над зоной другого цвета, а средний - над границами цветов. Гарантируется, что первые 150 мм границы линий - прямой участок.

Примеры полей:



Зона старта - зона размером 300x300 мм, центр которой совпадает с центром полигона и робота в момент старта.

Промежуточные зоны 1 и 2 - зоны размером 300x300 мм, центры которых совпадают с границей цветовых зон (промежуточные зоны расположены на прямых участках границы цветных зон).

Начисление баллов

Управляющая программа проверяется на пяти полигонах: одном известном заранее (из упражнений) и четырех неизвестных, имеющих разную конфигурацию цветовых зон. За каждое поле начисляется до 20% баллов. Полигон имеет следующие события, происходящие при совпадении следующих условий:

- "Out from start" - выводит сообщение в консоль "Out from start"; требуемые условия:
 - ни одна точка проекции робота не находится над зоной старта;
 - скорости левого и правого моторов ненулевые.
- "Intermediate 1 zone checker" - начисляет 5 баллов и выводит сообщение "The robot arrived at the first intermediate zone"; требуемые условия:
 - ранее наступило событие "Out from start";
 - центр проекции робота находится в промежуточной зоне 1.
- "Intermediate 2 zone checker" - начисляет 5 баллов и выводит сообщение "The robot arrived at the second intermediate zone"; требуемые условия:

- ранее наступило событие "Intermediate 1 zone checker";
- центр проекции робота находится в промежуточной зоне 2.
- "Finish checker" - начисляет 5 баллов и выводит сообщение в консоль "Successful finish"; завершает проверку; требуемые условия:
 - ранее наступило событие "Intermediate 2 zone checker";
 - центр робота находится в зоне старта.

Примеры заданий и тестовый проект

В приложенном архиве находятся следующие файлы:

- sen_task_12_exercise.qrs / sen_task_12_exercise_alt.qrs - файл-упражнение с настроенным тренировочным полем;
- sen_task_12.py - файл-программа с исходным кодом участника.

Файлы-упражнения могут использоваться для отладки программ. После отладки код программы необходимо перенести в файл sen_task_12.py и отправить в качестве решения задачи.

Решение

Задача похожа на предыдущую, но требует остановки после возврата в зону старта (после совершения полного круга). Так как полигон не имеет никаких явных обозначений старта или промежуточных зон, то роботу требуется определять свои координаты и понимать, где он находится. Это можно реализовать с помощью одометрии – определения положения робота и его частей в пространстве за счёт внутренних датчиков, то есть энкодеров моторов. Подробно о ней рассказано в видео [«Дифференциальная платформа роботов»](#). В текущей задаче можно воспользоваться полными формулами из видео или упрощёнными, предполагая, что при малых «шагах» (перемещениях робота) длина хорды примерно равна длине дуги.

За основу взято решение предыдущей задачи, у которого в цикл движения по границе зон добавлено условие завершения – цикл работает до тех пор, пока расстояние от центра робота до точки старте более 17,5 см (половина ширины / высоты зоны старта). А чтобы цикл не завершался сразу же на старте (так как робот изначально в нужной зоне), добавлен предварительный цикл с «обратным условием» - ехать по границе зон до тех пор, пока робот не отъедет на 30 см.

Расстояние до центра вычисляется по теореме Пифагора, как гипотенуза треугольника со сторонами, равными координатам X и Y робота (так как начальные координаты равны нулю).

Если в решении использовать упрощённые формулы одометрии, подразумевающие равенство хорд и дуг, то ошибка может накапливаться слишком быстро, в итоге робот может неточно определять свои координаты и останавливаться вне зоны старта, либо проезжать мимо нее. Чтобы улучшить результат вычисления можно добавить паузы в 30-50 мс на каждом шаге цикла. В итоге вычисления будут происходить реже, то есть ошибка будет накапливаться не так быстро.

Итоговый код программы может иметь следующий вид:

```
1. import sys
2. import time
3. import random
4. import math
5.
6. centrLine = brick.sensor("A4").read
```

```

7. leftLine = brick.sensor("A5").read
8. rightLine = brick.sensor("A6").read
9. leftMotor = brick.motor("M4")
10. rightMotor = brick.motor("M3")
11. leftEnc = brick.encoder("M4").read
12. rightEnc = brick.encoder("M3").read
13.
14. class Program():
15.     __interpretation_started_timestamp__ = time.time() * 1000
16.
17.     pi = 3.141592653589793
18.     Diam=0.056      # diameter of the robot's wheels, in meters
19.     L=0.077         # half-track of the robot, in meters
20.     X=-0.0475       # robot's start X coordinate, in meters
21.     Y=0.0           # robot's start Y coordinate, in meters
22.     orient = 0.0    # robot's orientation, in degrees or radians, at
the participant's discretion
23.     drivespd=50
24.     turnSpd=25
25.     kf=0.9
26.     kp_line=0.9
27.     ki_line=0
28.     kd_line=0
29.     errold_line=0
30.     I_line=0
31.
32.     left_light = 0
33.     right_light = 0
34.
35.     encL=0
36.     encR=0
37.
38.     Nleft_old=0
39.     Nright_old=0
40.
41.     def execMain(self):
42.         #add your code here
43.         self.X=-0.0475
44.         self.left_light = leftLine()
45.         self.right_light = rightLine()
46.         while math.sqrt( (self.X)**2 + (self.Y)**2)<0.3:
47.             self.LineTrace()
48.             print("Отъехали на 0.3 метра")
49.         while math.sqrt( (self.X)**2 + (self.Y)**2)>0.175:
50.             self.LineTrace()
51.             print("Вернулись на старт")
52.
53.         #motor stop on finish
54.         brick.motor("M3").poweroff()
55.         brick.motor("M4").powerOff()
56.         script.wait(10)
57.
58.         brick.stop()
59.         return
60.
61.     def LineTrace(self):
62.         errL = centrLine() - self.left_light
63.         errR = self.right_light - centrLine()
64.         err = abs(errL) - abs(errR)
65.         P=self.kp_line*err
66.         self.I_line=self.I_line+self.ki_line*err
67.         D=self.kd_line*(err-self.errold_line)
68.         U=P+self.I_line+D
69.         leftMotor.setPower(self.drivespd-U)
70.         rightMotor.setPower(self.drivespd+U)
71.         self.errold_line = err

```

```
72.         Nleft=leftEnc()
73.         Nright=rightEnc()
74.         dsleft = math.pi * self.Diam *(Nleft - self.Nleft_old)/360
75.         dsright = math.pi * self.Diam *(Nright - self.Nright_old)/360
76.         self.X = self.X+(dsright+dsleft)/2*math.cos( self.orient +
(dsrigh-dsleft)/(4*self.L) )
77.         self.Y = self.Y+(dsright+dsleft)/2*math.sin( self.orient +
(dsrigh-dsleft)/(4*self.L) )
78.         self.orient = self.orient+(dsright-dsleft)/(2*self.L)
79.         self.Nleft_old = Nleft
80.         self.Nright_old = Nright
81.         script.wait(50)
82.
83.     def main():
84.         program = Program()
85.         program.execMain()
86.
87.     if __name__ == '__main__':
88.         main()
```

Задача К. Циферблат

Задание

В симуляторе TRIK Studio подготовить управляющую программу, способную перемещать робота на заданное расстояние с выравниванием по гироскопу. Направление движения робот определяет, двигаясь по линии; но за метр до зоны финиша линия заканчивается.

Описание робота

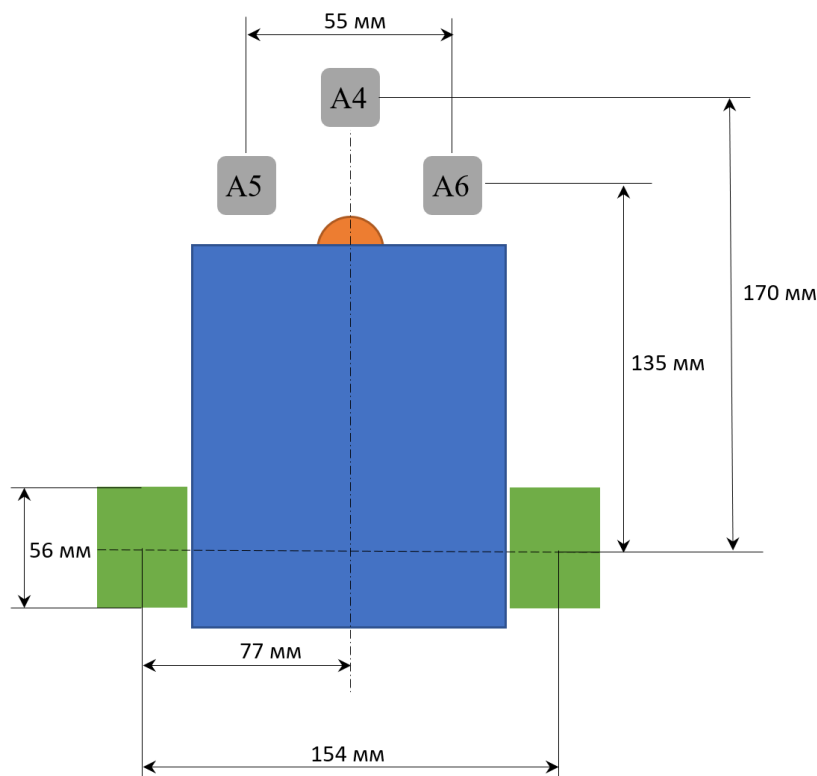
Дифференциальная платформа. Робот оснащен четырьмя датчиками:

К порту A5 подключен левый передний датчик.

К порту A6 подключен правый передний датчик.

К порту A4 подключен центральный 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.

Реалистичные физика, сенсоры и моторы включены.

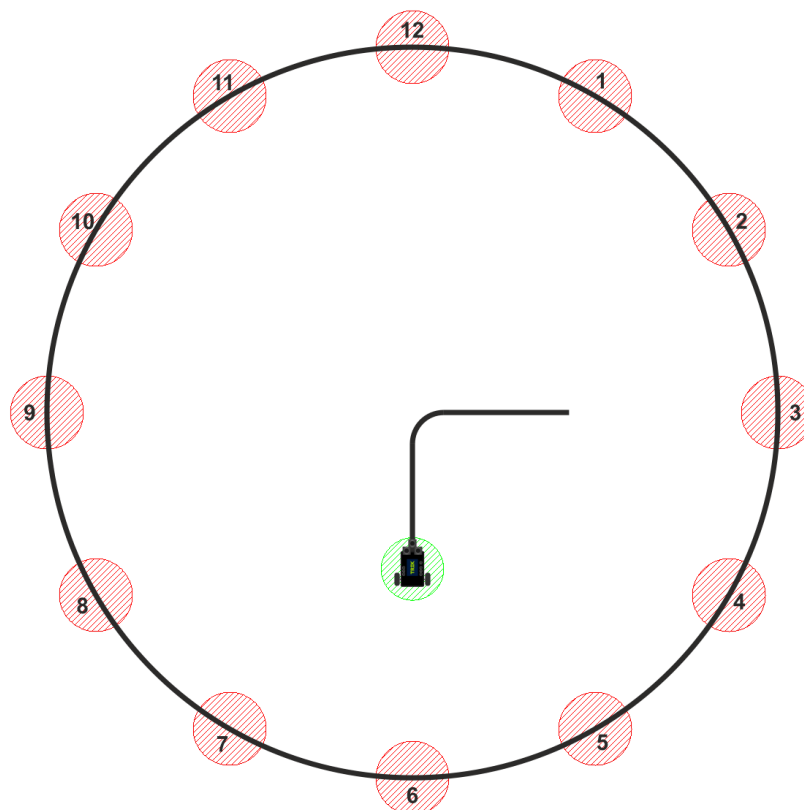
Полигон

Общий размер полигона составляет 4 x 4 метра.

Полигон визуально напоминает циферблат: линия образует "стрелки", а цифры циферблата - зоны остановки робота. Одна из "стрелок" направлена строго вниз, в направлении 6 часов. Вторая "стрелка" может быть направлена на любую цифру циферблата, кроме 6. Переход между "стрелками" образует дугу радиусом 150 мм. Ширина линии "стрелки" не менее 25 мм.

Гарантируется, что от конца второй "стрелки" до центра соответствующей зоны остановки - ровно 1 метр.

Пример полигона:



Зеленым обозначена зона старта - ее центр совпадает с центром робота на старте; размеры зоны 300x300 мм.

Красным обозначены зоны финиша - их центры совпадают с окружностью диаметром 3,5 м и расположены под углами 0, 30, 60 ... 330 градусов; размеры зон 350x350 мм.

Начисление баллов

Управляющая программа проверяется на пяти полигонах: одном известном заранее (из упражнения) и четырех неизвестных, имеющих разную конфигурацию стрелок. За каждое поле начисляется до 20% баллов. Полигон имеет следующие события, происходящие при совпадении следующих условий:

- "Out from start" - выводит сообщение в консоль "Out from start"; требуемые условия:
 - ни одна точка проекции робота не находится над зоной старта.
- "Successful finish checker" - начисляет 20 баллов и выводит сообщение в консоль "Successful finish"; завершает проверку; требуемые условия:
 - ранее наступило событие "Out from start";
 - центр робота находится в финишной зоне, на которую указывает вторая "стрелка";
 - скорости обоих моторов робота равны нулю.
- "Partial finish checker" - начисляет 10 баллов и выводит сообщение в консоль "Partial finish"; завершает проверку; требуемые условия:
 - ранее наступило событие "Out from start";

- любая точка проекции робота находится в финишной зоне, на которую указывает вторая "стрелка";
- скорости обоих моторов робота равны нулю.

Примеры заданий и тестовый проект

В приложенном архиве находятся следующие файлы:

- `sen_task_19_exercise.qrs` - файлы-упражнения с настроенными тренировочными полями;
- `sen_task_19.py` - файл-программа с исходным кодом участника.

Файлы-упражнения могут использоваться для отладки программ. После отладки код программы необходимо перенести в файл `sen_task_19.py` и отправить в качестве решения задачи.

Решение

Декомпозируем задачу на компоненты:

- Роботу необходимо двигаться по линии до тех пор, пока она не закончится.
- После окончания линии роботу необходимо продолжить движение в том же направлении на расстояние 1 метр.

Реализация движения по линии до ее окончания приведена в задаче Н (Стена). Движение на один метр можно реализовать с синхронизацией моторов, например, как в задаче Н. Но при конфигурации полигона со «стрелками» в направлении 5 и 7 часов длины «стрелки» может быть недостаточно для того, чтобы робот на ней эффективно выровнялся. В итоге робот может промахиваться мимо этих зон остановки. Для решения этой проблемы можно использовать встроенный в робота гироскоп.

Алгоритм решения задачи с использованием гироскопа следующий:

- До начала движения робот производит калибровку гироскопа и запоминает текущее направление как нулевое.
- Робот движется по линии до ее окончания.
- В момент окончания линии робот определяет зону, в направлении которой он сейчас движется. Например, если показания гироскопа от -15 до 15 градусов, то робот движется в направлении «12 часов», то есть в идеальном направлении 0 градусов; если показания гироскопа от 75 до 105 градусов, то робот движется в направлении «3 часа», то есть в идеальном направлении 90 градусов.
- Робот совершает проезд на 1 метр с регулировкой по гироскопу в идеальном направлении.

Регулятор по гироскопу аналогичен рассмотренным ранее регуляторам для движения по линии, вдоль стены и т.д. В качестве ошибки («невязки») регулятора берётся разница в показаниях гироскопа и идеальном направлении. Важно понимать, что гироскоп возвращает значения в виде миллиградусов, то есть их надо делить на 1000 перед сравнением с идеальным направлением

Итоговый код программы может иметь вид:

```
1. import sys
2. import time
```

```

3. import random
4. import math
5.
6. centrLine = brick.sensor("A4").read
7. leftLine = brick.sensor("A5").read
8. rightLine = brick.sensor("A6").read
9. leftMotor = brick.motor("M4")
10. rightMotor = brick.motor("M3")
11. leftEnc = brick.encoder("M4").read
12. rightEnc = brick.encoder("M3").read
13.
14. class Program():
15.     __interpretation_started_timestamp__ = time.time() * 1000
16.
17.     pi = 3.141592653589793
18.     Diam=0.056           #diameter of the robot's wheels, in meters
19.     L=0.077              #half-track of the robot, in meters
20.     drivespd=50
21.     turnSpd=25
22.     kf=0.9
23.     kp_line=1.5
24.     ki_line=0
25.     kd_line=0
26.     kp_enc=1.0
27.     ki_enc=0
28.     kd_enc=0
29.     errold_line=0
30.     errold_enc=0
31.     I_line=0
32.     I_enc=0
33.
34.     def execMain(self):
35.         # when calibrating in 2D, the robot's direction is remembered,
36.         from which the angle will be calculated.
37.         # The robot's initial direction is due north.
38.         # Lengthy 2D calibration is not required.
39.         brick.gyroscope().calibrate(200)
40.         script.wait(300)
41.
42.         #add your code here
43.
44.         self.LineToEnd()
45.         print("конец линии, переключение на синхронный поезд")
46.         moveAngle = self.GetCleanAngle()
47.         print("Угол движения = ", moveAngle )
48.         self.GyroToDist(moveAngle, 1.0)
49.         print("проехал заданное расстояние")
50.         #motor stop on finish
51.         brick.motor("M3").poweroff()
52.         brick.motor("M4").powerOff()
53.         script.wait(10)
54.
55.         brick.stop()
56.         return
57.
58.     def LineTrace(self):
59.         err = leftLine() - rightLine()
60.         P=self.kp_line*err
61.         self.I_line=self.I_line+self.ki_line*err
62.         D=self.kd_line*(err-self.errold_line)
63.         U=P+self.I_line+D
64.         leftMotor.setPower(self.drivespd-U)
65.         rightMotor.setPower(self.drivespd+U)
66.         self.errold_line = err
67.         script.wait(10)
68.         pass

```

```

68.
69.     def LineToEnd(self):
70.         while leftLine()+centrLine()+rightLine()>70:
71.             self.LineTrace()
72.             leftMotor.powerOff()
73.             rightMotor.powerOff()
74.             errold_line=0
75.             I_line=0
76.
77.     def SyncToDist(self, dist):
78.         brick.encoder("E3").reset()
79.         brick.encoder("E4").reset()
80.         pathAngle=0
81.         alpha = (360*dist)/(math.pi*self.Diam)
82.         while pathAngle < math.fabs(alpha):
83.             pathL= leftEnc()
84.             pathR=rightEnc()
85.             pathAngle = (pathL+pathR)/2
86.             self.SyncMove()
87.             leftMotor.powerOff()
88.             rightMotor.powerOff()
89.
90.     def SyncMove(self):
91.         err = leftEnc() - rightEnc()
92.         P=self.kp_enc*err
93.         self.I_enc=self.I_enc+self.ki_enc*err
94.         D=self.kd_enc*(err-self.errold_enc)
95.         U=P+self.I_enc+D
96.         rightMotor.setPower(self.drivespd + U)
97.         leftMotor.setPower(self.drivespd - U)
98.         self.errold_enc = err
99.         script.wait(10)
100.
101.     def GetCleanAngle(self):
102.         gyroData = brick.gyroscope().read()
103.         rawAngle = gyroData[6]/1000
104.         cleanAngle = 0
105.         if rawAngle>-180 and rawAngle<-135:
106.             cleanAngle = -150
107.         elif rawAngle>=-135 and rawAngle<-105:
108.             cleanAngle=-120
109.         elif rawAngle>=-105 and rawAngle<-75:
110.             cleanAngle=-90
111.         elif rawAngle>=-75 and rawAngle<-45:
112.             cleanAngle=-60
113.         elif rawAngle>=-45 and rawAngle<-15:
114.             cleanAngle=-30
115.         elif rawAngle>=-15 and rawAngle<15:
116.             cleanAngle=0
117.         elif rawAngle>=15 and rawAngle<45:
118.             cleanAngle=30
119.         elif rawAngle>=45 and rawAngle<75:
120.             cleanAngle=60
121.         elif rawAngle>=75 and rawAngle<105:
122.             cleanAngle=90
123.         elif rawAngle>=105 and rawAngle<135:
124.             cleanAngle=120
125.         elif rawAngle>=135 and rawAngle<180:
126.             cleanAngle=150
127.         return cleanAngle
128.
129.     def GyroMove(self, angle):
130.         gyroData = brick.gyroscope().read()
131.         rawAngle = gyroData[6]/1000
132.         err = rawAngle - angle
133.         P=self.kp_enc*err

```

```
134.         self.I_enc=self.I_enc+self.ki_enc*err
135.         D=self.kd_enc*(err-self.erold_enc)
136.         U=P+self.I_enc+D
137.         rightMotor.setPower(self.drivespd + U)
138.         leftMotor.setPower(self.drivespd - U)
139.         self.erold_enc = err
140.         script.wait(10)
141.
142.     def GyroToDist(self, angle, dist):
143.         brick.encoder("E3").reset()
144.         brick.encoder("E4").reset()
145.         pathAngle=0
146.         alpha = (360*dist)/(math.pi*self.Diam)
147.         while pathAngle < math.fabs(alpha):
148.             pathL= leftEnc()
149.             pathR=rightEnc()
150.             pathAngle = (pathL+pathR)/2
151.             self.GyroMove(angle)
152.             leftMotor.powerOff()
153.             rightMotor.powerOff()
154.
155.     def main():
156.         program = Program()
157.         program.execMain()
158.
159. if __name__ == '__main__':
160.     main()
```

Задача L. Сотовый лабиринт с линиями

Задание

В симуляторе TRIK Studio подготовить управляющую программу, способную перемещать робота по лабиринту из шестигранных элементов - "сот" (похожих на пчелиные соты). Робот начинает движение в центре лабиринта и должен добраться до соты с адресом E2. На полу полигона нанесена разметка в виде черных линий для облегчения навигации робота.

Описание робота

Дифференциальная платформа. Робот оснащен четырьмя датчиками:

К порту A5 подключен левый передний датчик.

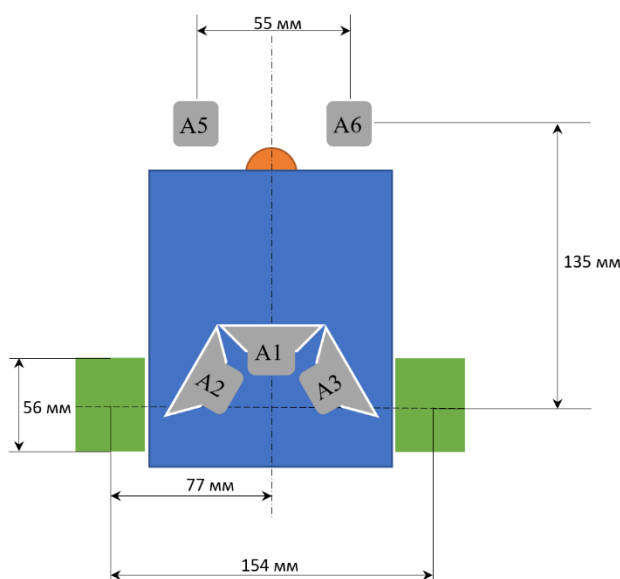
К порту A6 подключен правый передний датчик.

К порту A1 подключен датчик расстояния, направленный по направлению движения робота.

К порту A2 подключен датчик расстояния, направленный налево под 60 градусов относительно направления движения робота.

К порту A3 подключен датчик расстояния, направленный направо под 60 градусов относительно направления движения робота.

Размеры модели в симуляторе TRIK Studio представлены на рисунке.



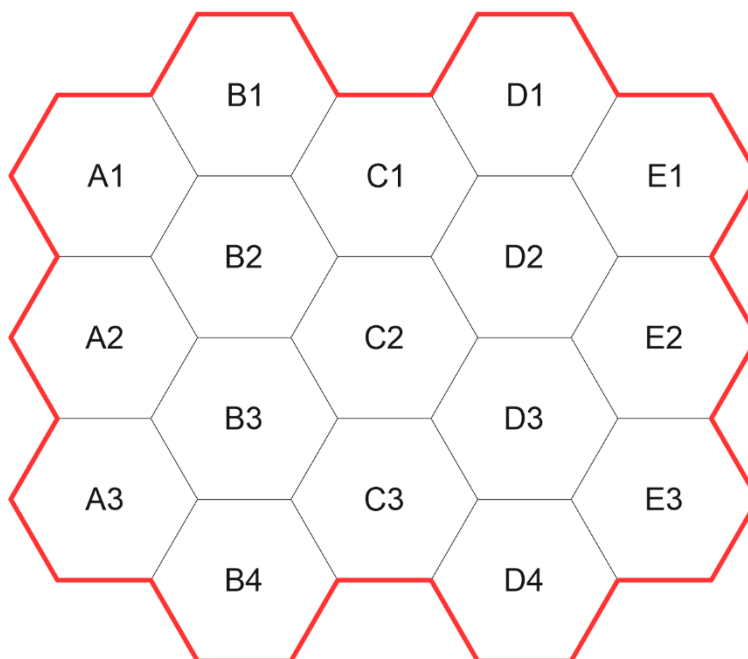
Левый мотор робота подключен к порту M4, правый мотор - к порту M3.

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

Реалистичные физика и моторы включены. Реалистичные сенсоры отключены.

Полигон

Общий размер полигона составляет 4 x 4 метра. На поле собран лабиринт из стен, образующих шестигранные соты. Размер каждой соты: 1000x866 мм.

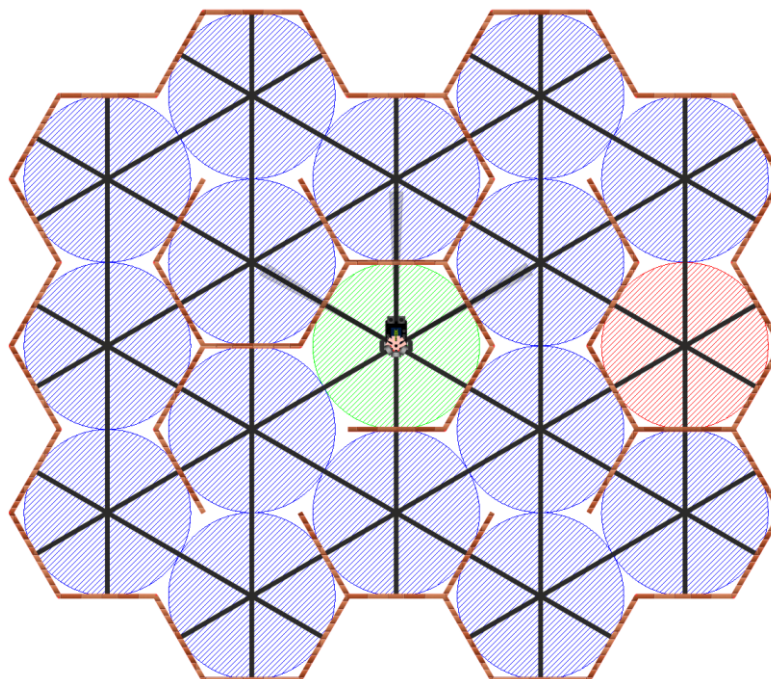


Робот начинает движение в центре полигона, в соте C2, в направлении вверх.

В каждой соте нанесена разметка из трех пересекающихся черных линий. Точка пересечения - в центре соты. Ширина линии не менее 25 мм. Линии нанесены даже если ведут к стене.

Гарантируется, что на старте линия расположена между датчиками A5 и A6. Гарантируется, что на старте центр оси колес робота в центре соты и датчики расстояния направлены ровно вдоль линий.

Пример полигона:



Зеленым обозначена зона старта - ее центр совпадает с соты C2; размеры зоны 866x866 мм.

Красным обозначена зона финиша - ее центр совпадает с центром соты E2; размеры зоны 866x866 мм.

Синим обозначены промежуточные зоны - во всех сотах, кроме C2 и E2, размеры зон 866x866 мм, а центры совпадают с центрами сот.

Начисление баллов

Управляющая программа проверяется на пяти полигонах: одном известном заранее (из упражнения) и четырех неизвестных, имеющих разную конфигурацию стен. За каждое поле начисляется до 20% баллов. Полигон имеет следующие события, происходящие при совпадении следующих условий:

- "Out from start" - выводит сообщение в консоль "Out from start"; требуемые условия:
 - ни одна точка проекции робота не находится над зоной старта.
- "XY-checker" - 15 событий для 15 промежуточных зон, каждая начисляет по 1 баллу и выводит сообщение об этом в консоль; требуемые условия:
 - все точки проекции робота находятся в соответствующей промежуточной зоне;
 - скорости обоих моторов робота равны нулю;
 - последнее выведенное на экран робота сообщение совпадает с названием соты промежуточной зоны (например: A1, B2, D4 и т.д.).
- "Finish checker" - начисляет баллы до 20 (то есть итоговый балл будет ровно 20, независимо от набранных ранее баллов за промежуточные зоны) и выводит сообщение в консоль "Successful finish"; завершает проверку; требуемые условия:
 - все точки проекции робота находятся в финишной зоне;
 - скорости обоих моторов робота равны нулю;
 - последнее выведенное на экран робота сообщение было "E2" (без кавычек).
- "Not finish checker" - выводит сообщение в консоль "Not zone finish"; завершает проверку; требуемые условия:
 - ни одна точка проекции робота не находится в финишной зоне;
 - скорости обоих моторов робота равны нулю;
 - последнее выведенное на экран робота сообщение было "E2" (без кавычек).

Примеры заданий и тестовый проект

В приложенном архиве находятся следующие файлы:

- sen_task_5_exercise.qrs - файл-упражнение с настроенным тренировочным полем;
- sen_task_5.py - файл-программа с исходным кодом участника.

Файлы-упражнения могут использоваться для отладки программ. После отладки код программы необходимо перенести в файл sen_task_5.py и отправить в качестве решения задачи.

Решение

Задача подразумевает движение по лабиринту, но нестандартной формы. Для прохода лабиринта можно использовать алгоритм одной руки – «правило правой руки» или «правило левой руки». Полный обход лабиринта может занять много времени (более 180 секунд, отведённых на прохождение полигона), поэтому базовые скорости движения по линии и поворотов, возможно, придётся увеличить для прохождения всех лабиринтов.

Дальнейшее решение будет основано на «правиле правой руки».

В каждой новой «соте» требуется определить путь к следующей «соте». Для этого можно проверить правый датчик расстояния, и если он показывает более 70 см (то есть, проезд в следующую «соту»), то следует повернуться направо до линии и ехать до перекрёстка – в следующую соту. Если же правый датчик видит менее 70 см, то следует проверить передний датчик расстояния и проехать в следующую соту, если там есть проезд. А если и передний датчик не видит проезда, то надо поворачиваться налево и проверять передний датчик до тех пор, пока не найдем проезд.

Каждый проезд в следующую «соту» должен выполняться по следующему алгоритму:

- проезд по линии до перекрёстка;
- проезд по линии на 14,5 см (чтобы центр оси колёс попал на перекрёсток и удобнее было поворачивать);
- поворот направо до линии, чтобы следующий поиск проезда не пропустил проезды справа.

После реализации проезда лабиринта по правилу одной руки требуется научить робота определять текущие координаты. Каждый раз, выполняя проезд из «соты» в «соту» необходимо их обновлять – добавлять или вычитать по единице из координат X и Y робота. Буквы в названии «сот» будут координатой X, цифры – координатой Y. Причём эти координаты будут изменяться по разным правилам в зависимости от направления робота и от чётности / нечётности координаты X. Подробно этот алгоритм можно увидеть в итоговом коде программы:

```

1. import sys
2. import time
3. import random
4. import math
5.
6. leftIK = brick.sensor("A2").read
7. centrik = brick.sensor("A1").read
8. rightIK = brick.sensor("A3").read
9. leftLine = brick.sensor("A5").read
10. rightLine = brick.sensor("A6").read
11. leftMotor = brick.motor("M4")
12. rightMotor = brick.motor("M3")
13. leftEnc = brick.encoder("M4").read
14. rightEnc = brick.encoder("M3").read
15.
16. class Program():
17.     __interpretation_started_timestamp__ = time.time() * 1000
18.
19.     pi = 3.141592653589793
20.     Diam=0.056 #диаметр колес робота, в метрах
21.     L=0.077 #полуколея робота, в метрах
22.
23.     x=-0.0475 #координата X робота, в метрах
24.     y=0.0 #координата Y робота, в метрах
25.     orient = 0.0 #ориентация робота, в градусах или радианах на
    усмотрение участника
26.     #на старте все координаты робота нулевые
27.     #нулевая ориентация означает направление вдоль оси OX
28.     #во время движения рекомендуется обновлять эти переменные
29.     cellx=3
30.     celly=2
31.     cellorient=0
32.
33.     drivespd=80
34.     turnspd=40
35.     kf=0.9
36.     kp_line=0.7

```

```

37.         ki_line=0.15
38.         kd_line=0
39.         kp_wall=1.0
40.         ki_wall=0.05
41.         kd_wall=0.1
42.         kp_enc=1.0
43.         ki_enc=0
44.         kd_enc=0
45.         errold_line=0
46.         errold_wall=0
47.         errold_enc=0
48.         I_line=0
49.         I_wall=0
50.         I_enc=0
51.
52.         encL=0
53.         encR=0
54.
55.         Nleft_old=0
56.         Nright_old=0
57.
58.     def execMain(self):
59.         #добавьте свой код ниже
60.
61.         while (self.cellX != 5) or (self.cellY != 2):
62.             self.PrintCellCoordinate()
63.             leftMotor.powerOff()
64.             rightMotor.powerOff()
65.             script.wait(10)
66.             self.RightHandMove()
67.
68.             self.PrintCellCoordinate()
69.             print("Finish")
70.
71.             #остановка моторов на финише
72.             brick.motor("M3").powerOff()
73.             brick.motor("M4").powerOff()
74.             script.wait(10)
75.
76.             brick.stop()
77.             return
78.
79.         #поиск пути в след клетку по правилу правой руки
80.         #по окончании робот должен стоять в направлении след клетки
81.     def FindRightPath(self):
82.         if rightIK()>70:
83.             #поворот направо до линии
84.             self.RotateRightToLine()
85.             self.cellorient=(self.cellorient+1)%6
86.         elif centrIK()>70:
87.             #проезд в след клетку
88.             pass
89.         else:
90.             #поворот налево, пока не найдем проезд
91.             while centrIK() < 70:
92.                 self.RotateLeftToLine()
93.                 self.cellorient=(self.cellorient-1)%6
94.             pass
95.
96.         #проезд в след клетку по правилу правой руки
97.         #после проезда поворот направо на одну линию
98.     def MoveToNextCellRight(self):
99.         #проезд в след клетку
100.        self.LineToCrossroad()
101.        self.LineToDist(0.14)
102.        self.UpdateCellCoordinate()

```

```
103.         #поворот направо на одну линию
104.         self.RotateRightToLine()
105.         self.cellorient=(self.cellorient+1)%6
106.
107.     def RightHandMove(self):
108.         self.FindRightPath()
109.         self.MoveToNextCellRight()
110.
111.     def PrintCellCoordinate(self):
112.         letter = 'Z'
113.         if self.cellx == 1:
114.             letter = 'A'
115.         if self.cellx == 2:
116.             letter = 'B'
117.         if self.cellx == 3:
118.             letter = 'C'
119.         if self.cellx == 4:
120.             letter = 'D'
121.         if self.cellx == 5:
122.             letter = 'E'
123.         addr = letter+str(self.celly)
124.         print(addr)
125.         brick.display().addLabel(addr, 20, 20, 20)
126.         brick.display().redraw()
127.
128.     def UpdateCellCoordinate(self):
129.         if self.cellorient == 0:
130.             self.cellx += 0
131.             self.celly += -1
132.         if self.cellorient == 1:
133.             if self.cellx%2 == 0:
134.                 self.celly += -1
135.             else:
136.                 self.celly += 0
137.                 self.cellx += 1
138.         if self.cellorient == 2:
139.             if self.cellx%2 == 0:
140.                 self.celly += 0
141.             else:
142.                 self.celly += 1
143.                 self.cellx += 1
144.         if self.cellorient == 3:
145.             self.cellx += 0
146.             self.celly += 1
147.         if self.cellorient == 4:
148.             if self.cellx%2 == 0:
149.                 self.celly += 0
150.             else:
151.                 self.celly += 1
152.                 self.cellx += -1
153.         if self.cellorient == 5:
154.             if self.cellx%2 == 0:
155.                 self.celly += -1
156.             else:
157.                 self.celly += 0
158.                 self.cellx += -1
159.
160.     def RotateLeftToLine(self):
161.         rightMotor.setPower(self.turnSpd)
162.         leftMotor.setPower(-self.turnSpd)
163.         moveTime=0
164.         startTime = time.time() * 1000
165.         while moveTime<500:
166.             moveTime = time.time() * 1000 - startTime
167.             script.wait(20)
168.         while rightLine()<30:
```

```

169.         script.wait(20)
170.         leftMotor.powerOff()
171.         rightMotor.powerOff()
172.         brick.playTone(1000, 100)
173.
174.     def RotateRightToLine(self):
175.         rightMotor.setPower(-self.turnSpd)
176.         leftMotor.setPower(self.turnSpd)
177.         moveTime=0
178.         startTime = time.time() * 1000
179.         while moveTime<500:
180.             moveTime = time.time() * 1000 - startTime
181.             script.wait(20)
182.         while leftLine()<30:
183.             script.wait(20)
184.             leftMotor.powerOff()
185.             rightMotor.powerOff()
186.             brick.playTone(1000, 100)
187.
188.     def LineTrace(self):
189.         err = leftLine() - rightLine()
190.         P=self.kp_line*err
191.         self.I_line=self.I_line+self.ki_line*err
192.         D=self.kd_line*(err-self.erold_line)
193.         U=P+self.I_line+D
194.         leftMotor.setPower(self.drivespd-U)
195.         rightMotor.setPower(self.drivespd+U)
196.         self.erold_line = err
197.         script.wait(20)
198.
199.     def LineToCrossroad(self):
200.         while ( leftLine() + rightLine() )<60:
201.             self.LineTrace()
202.             leftMotor.powerOff()
203.             rightMotor.powerOff()
204.
205.     def SyncToDist(self, dist):
206.         brick.encoder("E3").reset()
207.         brick.encoder("E4").reset()
208.         pathAngle=0
209.         alpha = (360*dist)/(math.pi*self.Diam)
210.         while pathAngle < math.fabs(alpha):
211.             pathL= leftEnc()
212.             pathR=rightEnc()
213.             pathAngle = (pathL+pathR)/2
214.             self.SyncMove()
215.             leftMotor.powerOff()
216.             rightMotor.powerOff()
217.
218.     def SyncMove(self):
219.         err = leftEnc() - rightEnc()
220.         P=self.kp_enc*err
221.         self.I_enc=self.I_enc+self.ki_enc*err
222.         D=self.kd_enc*(err-self.erold_enc)
223.         U=P+self.I_enc+D
224.         rightMotor.setPower(self.drivespd + U)
225.         leftMotor.setPower(self.drivespd - U)
226.         self.erold_enc = err
227.         script.wait(20)
228.
229.     def LineToDist(self, dist):
230.         startEncL = leftEnc()
231.         startEncR = rightEnc()
232.         pathAngle=0
233.         alpha = (360*dist)/(math.pi*self.Diam)
234.         while pathAngle < math.fabs(alpha):

```

```
235.         pathL=leftEnc()-startEncL
236.         pathR=rightEnc()-startEncR
237.         pathAngle = (pathL+pathR)/2
238.         self.LineTrace()
239.         leftMotor.powerOff()
240.         rightMotor.powerOff()
241.
242.     def main():
243.         program = Program()
244.         program.execMain()
245.
246.     if __name__ == '__main__':
247.         main()
```

После реализации этих алгоритмов достаточно настроить вывод координат на экран робота и в консоль (для удобства отладки). И настроить условие для цикла, выполняющего поиск пути по правилу правой руки – повторять его до тех пор, пока координаты робота не станут E2 (то есть X=5, Y=2).