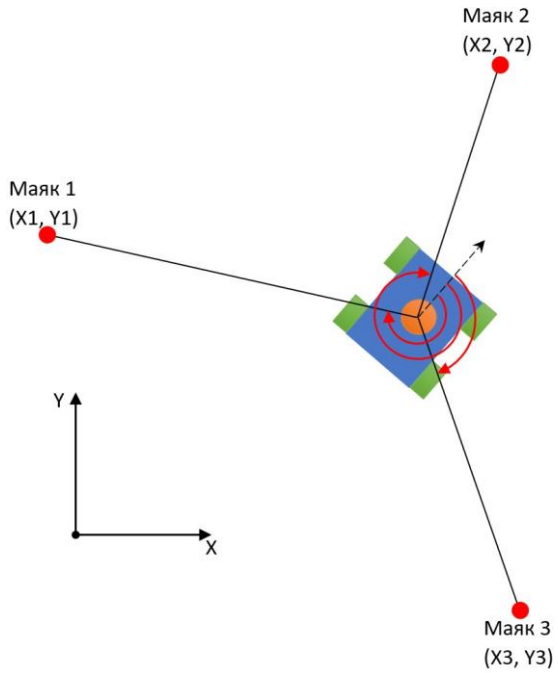
Оглавление

Первая часть задач

Задача на локализацию по маякам и пеленгам (1.1.1)

На полигоне находится три "маяка" с известными координатами. Робот оборудован устройством, способным определить направления на маяки. При этом азимут измеряется относительно направления движения робота вращением по часовой стрелке. То есть, маяк, расположенный ровно по направлению движения робота имеет азимут 0 градусов; маяк, расположенный строго справа от робота - азимут 90 градусов; маяк, расположенный строго слева от робота - азимут 270 градусов (не -90 градусов!).

Даны координаты "маяков" и азимуты на маяки относительно направления движения робота. Необходимо определить координаты робота и его направление. Допустимая погрешность +- 1 мм и +- 1 градус.



Структура позиций объектов в задаче

Формат входных данных

Первые 3 строки содержат через пробел по два целых числа — xi , yi — координаты маяков в миллиметрах ($xi \in [-5000; 5000]$, $yi \in [-5000; 5000]$).

Четвертая строка содержит через пробел три целых числа — $a1$, $a2$, $a3$ — азимуты на маяки относительно направления движения робота ($ai \in [0; 359]$).

Формат выходных данных

Строка, содержащая через пробел три числа: координату X робота в миллиметрах, координату Y робота в миллиметрах, угол в градусах между осью X системы координат и направлением движения робота.

Комментарии

Готовые библиотеки для решения задачи использовать запрещено. Примеры входных данных представлены по ссылке и не выводятся в самом задании.

Примеры

Примеры тестовых полигонов представлены по данной ссылке.

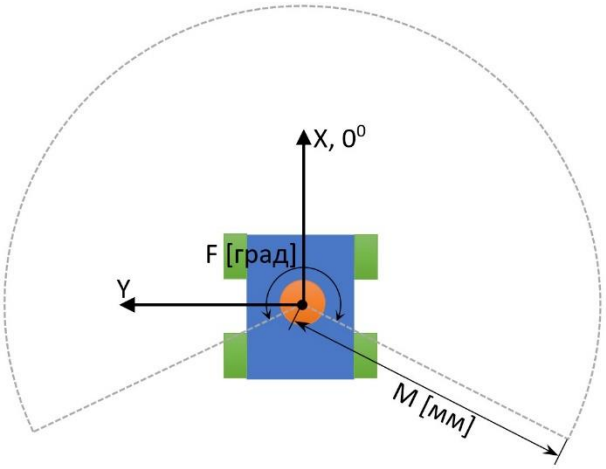
Решение

Псевдокод решения:

```
function calculateRobotPositionAndDirection(data):
    beacon1 = {x: data[0][0], y: data[0][1]}
    beacon2 = {x: data[1][0], y: data[1][1]}
    beacon3 = {x: data[2][0], y: data[2][1]}
    azimuths = data[3]

    // Calculate the distances between the beacons
    dist12 = calculateDistance(beacon1, beacon2)
    dist13 = calculateDistance(beacon1, beacon3)
    dist23 = calculateDistance(beacon2, beacon3)

    // Calculate the angles between the beacons using the cosine theorem
    angle1 = calculateAngle(dist12, dist13, dist23)
    angle2 = calculateAngle(dist12, dist23, dist13)
    angle3 = calculateAngle(dist13, dist23, dist12)

    // Determine the direction of the robot
    robotDirection = calculateRobotDirection(azimuths)

    // Calculate the robot's position based on beacon coordinates
    robotPosition = calculateRobotPosition(beacon1, beacon2, beacon3, angle1, angle2, angle3)

    return robotPosition + " " + robotDirection


function calculateDistance(point1, point2):
    // Calculate the Euclidean distance between two points
    distance = sqrt((point2.x - point1.x)^2 + (point2.y - point1.y)^2)
    return distance


function calculateAngle(side1, side2, side3):
    // Calculate the angle using the cosine theorem
    numerator = side1^2 + side2^2 - side3^2
    denominator = 2 * side1 * side2
    angle = acos(numerator / denominator)
    return angle


function calculateRobotDirection(azimuths):
    // Determine the direction of the robot based on azimuths
    direction = (azimuths[0] + azimuths[1] + azimuths[2]) / 3
    return direction


function calculateRobotPosition(beacon1, beacon2, beacon3, angle1, angle2, angle3):
    // Calculate the robot's position based on beacon coordinates and angles
    x = (beacon1.x + beacon2.x + beacon3.x) / 3
    y = (beacon1.y + beacon2.y + beacon3.y) / 3
    angle = calculateRobotAngle(angle1, angle2, angle3)

    return x + " " + y + " " + angle


function calculateRobotAngle(angle1, angle2, angle3):
    // Calculate the angle between the X axis and the robot's movement direction
    angle = (angle1 + angle2 + angle3) / 3
    return angle
```

```
// Example usage
data = [
    [1000, 2000],
    [1500, 2500],
    [1200, 1800],
    [45, 90, 135]
]
result = calculateRobotPositionAndDirection(data)
print("Robot position and direction:", result)
```

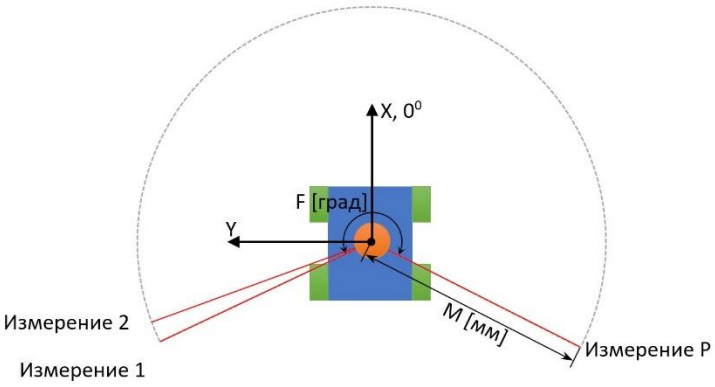
Задача на перевод координат (1.1.2)



Изображение системы

Робот оборудован LIDAR'ом. Угол его сканирования F градусов, на эти F градусов приходится P измерений LIDAR'а. Максимальное расстояние измерения - до M миллиметров. Если LIDAR не видит объекта, то соответствующее измерение имеет максимальное значение. Нулевые координаты декартовой системы координат OXY совпадают с центром LIDAR'а, от которого он измеряет расстояние. Ось X этой системы координат совпадает с серединой сектора сканирования LIDAR'а. Кроме того, в системе присутствует полярная система координат. Ее полюс совпадает с началом декартовой системы координат и центром лидара. А полярная ось (нулевой луч) совпадает с осью X декартовой системы координат.

Необходимо перевести координаты каждой найденной точки объекта в декартовы. При этом измерения LIDAR'а, которые не обнаружили объект, переводить в декартовы координаты не требуется.



Изображение условия задачи

Формат входных данных

Первая строка содержит целое число — F — угол в градусах ($F \in [1; 360]$).

Вторая строка содержит целое число — P — количество измерений ($P \in [1; 4096]$).

Третья строка содержит целое число — M — дистанция измерения LIDAR'а в миллиметрах ($M \in [1; 4000]$).

Четвертая строка содержит P целых чисел - показания LIDAR'a.

Формат выходных данных

Первая строка содержит целое число - количество измерений LIDAR'a, в которых обнаружен объект. Если ни в одном измерении объекты не обнаружены, то строка содержит число 0. Следующие строки присутствуют только если в первой число больше нуля; они содержат через пробел пары чисел с плавающей точкой - координаты X и Y точки обнаруженного объекта с точностью до 0.1 мм. Вторая строка содержит координаты из первого измерения лидара, в котором обнаружен объект, третья - из второго измерения, в котором обнаружен объект и т.д.

Комментарии

Готовые библиотеки для решения задачи использовать запрещено. Примеры входных данных представлены по ссылке и не выводятся в самом задании.

Примеры

Примеры тестовых полигонов представлены по данной ссылке.

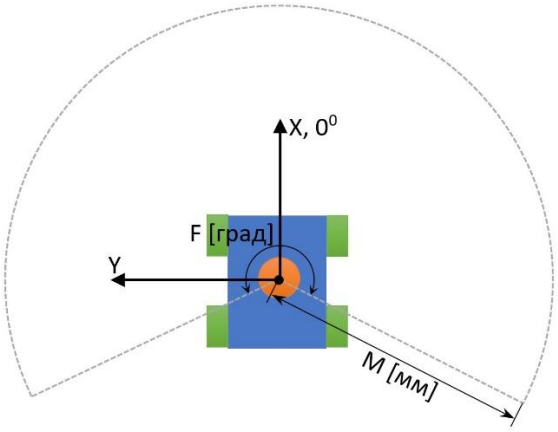
Решение

Псевдокод решения:

```
from math import cos, radians, sin, pi

angle = int(input())
num_dimensions = int(input())
distance = int(input())
lidar_readings = list(map(int, input().split()))

result_dimensions = 0
detected_points = []

for i in range(num_dimensions):
    if lidar_readings[i] != 0:
        result_dimensions += 1
        x_coord = lidar_readings[i] * cos(radians(angle / 2 - distance * i))
        y_coord = lidar_readings[i] * sin(pi - radians(angle / 2 - distance * i))
        x_coord = round(x_coord, 1)
        y_coord = round(y_coord, 1) * (-1)
        detected_points.append((x_coord, y_coord))

print(result_dimensions)
if result_dimensions > 0:
    for point in detected_points:
        x_val, y_val = point
        print("X:", x_val, "Y:", y_val)
```

Задача на SLAM (1.1.3)

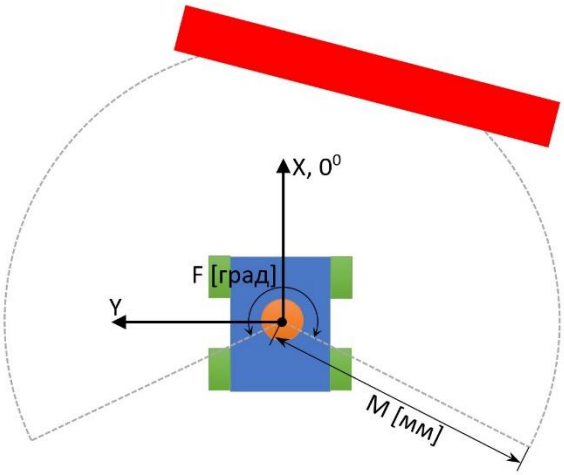
Продолжение предыдущей задачи(1.1.2)



Изображение системы

Робот оборудован LIDAR'ом. Угол его сканирования F градусов, на эти F градусов приходится P измерений LIDAR'а. Максимальное расстояние измерения - до M миллиметров. Если LIDAR не видит объекта, то соответствующее измерение имеет максимальное значение. Нулевые координаты декартовой системы координат OXY совпадают с центром LIDAR'а, от которого он измеряет расстояние. Ось X этой системы координат совпадает с серединой сектора сканирования LIDAR'а. Кроме того, в системе присутствует полярная система координат. Ее полюс совпадает с началом декартовой системы координат и центром лидара. А полярная ось (нулевой луч) совпадает с осью X декартовой системы координат.

Гарантируется, что в зоне видимости лидара находится одна стена без изгибов или углов. Необходимо определить угол между осью X декартовой системы координат и стеной.



Изображение условия задачи

Формат входных данных

Первая строка содержит целое число — F — угол в градусах ($F \in [1; 360]$).

Вторая строка содержит целое число — P — количество измерений ($P \in [1; 4096]$).

Третья строка содержит целое число — M — дистанция измерения LIDAR'а в миллиметрах ($M \in [1; 4000]$).

Четвертая строка содержит P целых чисел - показания LIDAR'а.

Формат выходных данных

Целое число - угол в градусах между стеной и осью X (направление движения робота).

Комментарии

Готовые библиотеки для решения задачи использовать запрещено. Примеры входных данных представлены по ссылке и не выводятся в самом задании.

Математическая система совпадает с задачей 1.1.2

Примеры

Примеры тестовых полигонов представлены по данной ссылке.

Решение

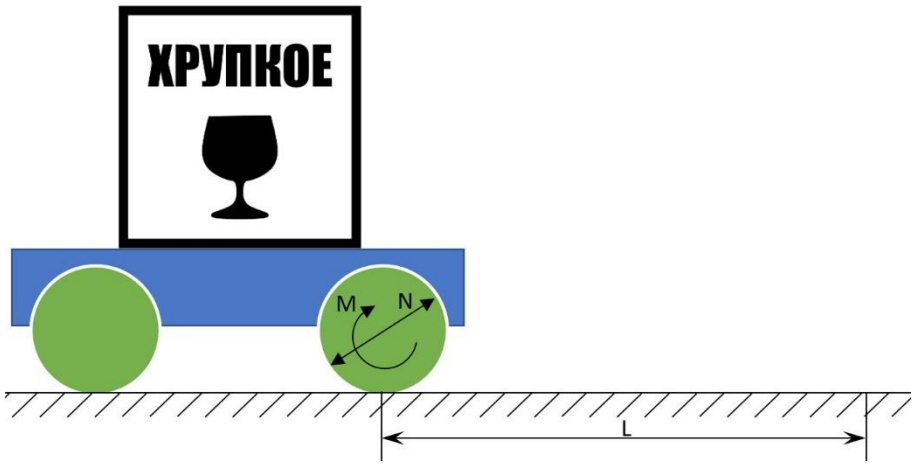
Псевдокод решения:

```
Read angle F from input
Read number of dimensions P from input
Read lidar measurement distance M from input
Read lidar readings from input and store them in an array or list

# Convert the angle from degrees to radians
angle_rad = F * pi / 180

# Initialize variables for wall distance and wall angle
wall_distance = infinity
wall_angle = 0

# Iterate through the lidar readings
for i = 0 to P-1:
    # Calculate the distance from the lidar reading
    distance = M * sin(angle_rad)

    # Update the wall distance and angle if a closer wall is found
    if distance < wall_distance:
        wall_distance = distance
        wall_angle = angle_rad

    # Increment the angle for the next lidar reading
    angle_rad += (360 / P) * pi / 180

# Convert the wall angle from radians to degrees
wall_angle_deg = wall_angle * 180 / pi

# Output the wall angle
Print wall_angle_deg
```

Задача на управление мотором по определенному закону (1.1.4)

Робот оборудован колесами диаметром N мм. Его приводы способны развивать скорость вращения колес до M $\text{обмин}_{\text{миноб}}$. Робот перевозит хрупкий груз, который разрушается при ускорении или торможении выше P $\text{мс}2_{\text{с2м}}$. Гарантируется, что приводы могут развить ускорение робота выше P $\text{мс}2_{\text{с2м}}$. Расстояние перемещения робота - L метров.

Необходимо вычислить минимальное время проезда робота T и скорости робота через каждые 0.1 секунды.



Изображение системы

Формат входных данных

Первая строка содержит целое число — N — диаметр колес в миллиметрах ($N \in [1; 250]$).

Вторая строка содержит целое число — M — максимальная скорость приводов в оборотах в минуту ($M \in [1; 400]$).

Третья строка содержит число с плавающей точкой — P — максимально допустимое ускорение / торможение, при котором разрушается груз ($P \in [0.001; 5.0]$).

Четвертая строка содержит число с плавающей точкой и двумя знаками после запятой — L — расстояние, на которое необходимо перевезти груз (в метрах) ($L \in [0.01; 100.00]$).

Формат выходных данных

T*10+1 строк.

Первая строка содержит минимальное время проезда в секундах, с точностью до 0.1 секунды. Следующие T*10 строк содержат скорости робота в каждый интервал времени в м/с (с точностью до 0.001 м/с). При этом вторая строка содержит скорость в момент старта, то есть число 0 м/с, третья - скорость в момент времени 0.1 с, четвертая - скорость в момент времени 0.2 с и т.д. Последняя строка содержит скорость робота в момент времени T с, то есть 0 м/с.

Комментарии

Готовые библиотеки для решения задачи использовать запрещено. Примеры входных данных представлены по ссылке и не выводятся в самом задании.

Примеры

Примеры тестовых полигонов представлены по данной ссылке.

Решение

Код решения на языке Python от одного из участников:

```
import math

d = float(input())
d /= 1000
m = float(input())
a_mx = float(input())
l = float(input())
vmx = ((math.pi * d * m) / 60)
dv = a_mx / 10
sqv = a_mx * l
ans = []
#print(vmx, sqv)
if sqv <= vmx * vmx:
    #print(sqv, vmx)
    #print("#1")
    v = math.sqrt(sqv)
    #print("&", v)
    t1 = v / a_mx
    #print("t1", t1)
    t = 0
    while t <= t1:
        v = a_mx * t
        t += 0.1
        ans.append(round(v, 3))
    v = a_mx * t1
    va = v
    while (t < 2 * t1) and v != 0:
        v = va - a_mx * (t - t1)
        t+=0.1
        ans.append(round(v, 3))
else:
    t1 = vmx / a_mx
    t2 = (l - vmx * t1) / vmx
    t = 0
    v = 0
    while t < t1:
        v = a_mx * t
        t+=0.1
        ans.append(round(v, 3))
    while t < t1 + t2:
        v = vmx
        ans.append(round(v, 3))
        t+=0.1
    while (t < 2 * t1 + t2) and (v != 0):
        v = vmx - a_mx * (t - t1 - t2)
        ans.append(round(v, 3))
        t+=0.1
#print(vmx)
print(len(ans) * 0.1)
for i in ans:
    print(i)
print(0)
```

Задача на компьютерное зрение (1.1.5)

Есть изображение с высотой H пикселей и шириной W пикселей. Необходимо вывести наиболее встречающийся в картинке цвет из списка: (RED, YELLOW, GREEN, BLUE, PINK, WHITE, BLACK)

Схема перевода из наиболее встречающегося цвета в формате HSV в текстовый вариант можно найти по данной ссылке. Файл содержит 18 строк, каждая из которых представляет собой границы H S V от начального значения включительно и до конечного включительно, которые нужно проверить по порядку. Первое попадание в границы HSV из файла и будет обозначать конечный цвет.

H[Hue] - ($H \in [0; 359]$) S[Saturation] - ($S \in [0; 255]$) V[Value] - ($V \in [0; 255]$)

```
0 20 60 255 125 255 'RED'
```

=> RED

0 <= H <= 20 and 60 <= S <= 255 and 125 <= V <= 255

Пример условия для перевода из HSV в текст

Пример простого изображения

В данном изображении наиболее встречающийся цвет красный(RED)

Формат входных данных

Первая строка содержит 2 целых числа — H и W — высота и ширина изображения в пикселях ($H \in [50; 4000]$, $W \in [50; 4000]$).

Вторая строка содержит через пробел H на W кодов цвета в шестнадцатеричной системе счисления. Код выглядит как $RRGGBB$, где RR это шестнадцатеричное число описывающее значение красного канала в пикселе (при переводе в десятичную оно принимает значения [0, 255]), GG аналогично описывает зеленый канал в пикселе, а BB описывает синий.

Формат выходных данных

Одно слово - наиболее встречающийся цвет из списка: (RED, YELLOW, GREEN, BLUE, PINK, WHITE, BLACK)

Комментарии

Готовые библиотеки для решения задачи использовать запрещено. Решать задачу методом подбора запрещено.

Примеры входных данных представлены по ссылке и не выводятся в самом задании.

В данной задаче вам специально дано 5 открытых картинок с ответами, чтобы вы смогли довести алгоритм до идеала. Алгоритм нужно оптимизировать на основе этих изображений.

Примеры

Примеры тестовых полигонов представлены по данной ссылке.

Решение

Код решения на языке Python:

```
import numpy as np
import os
import matplotlib.pyplot as plt


# find most common color in the image
def centroid_histogram(clt):
    # grab the number of different clusters and create a histogram
    # based on the number of pixels assigned to each cluster
    numLabels = np.arange(0, len(np.unique(clt.labels_)) + 1)
    (hist, _) = np.histogram(clt.labels_, bins=numLabels)
    # normalize the histogram, such that it sums to one
    hist = hist.astype("float")
    hist /= hist.sum()
    # return the histogram
    return hist


def plot_colors(hist, cluster_centers_):
    # initialize the bar chart representing the relative frequency
    # of each of the colors
    bar = np.zeros((50, 300, 3), dtype="uint8")
    startX = 0
    # loop over the percentage of each cluster and the color of
    # each cluster
    for (percent, color) in zip(hist, cluster_centers_):
        # plot the relative percentage of each cluster
        endX = startX + (percent * 300)
        import cv2
        cv2.rectangle(bar, (int(startX), 0), (int(endX), 50),
                      color.astype("uint8").tolist(), -1)
        startX = endX
    # return the bar chart
    return bar


class KMeans:
    def __init__(self, n_clusters=8, max_iter=300, tol=1e-4):
        self.n_clusters = n_clusters
        self.max_iter = max_iter
        self.tol = tol

    def fit(self, X):
        self.labels_ = np.zeros(X.shape[0])
        self.cluster_centers_ = np.random.uniform(X.min(), X.max(), size=(self.n_clusters, X.shape[1]))
        for tmp_i in range(self.max_iter):
            self.labels_ = np.argmin(np.linalg.norm(X[:, np.newaxis, :] - self.cluster_centers_, axis=2), axis=1)
            new_centers = np.array([X[self.labels_ == clus].mean(axis=0) for clus in range(self.n_clusters)])
            if np.linalg.norm(self.cluster_centers_ - new_centers) < self.tol:
                break
            self.cluster_centers_ = new_centers

    def predict(self, X):
        return np.argmin(np.linalg.norm(X[:, np.newaxis, :] - self.cluster_centers_, axis=2), axis=1)
```

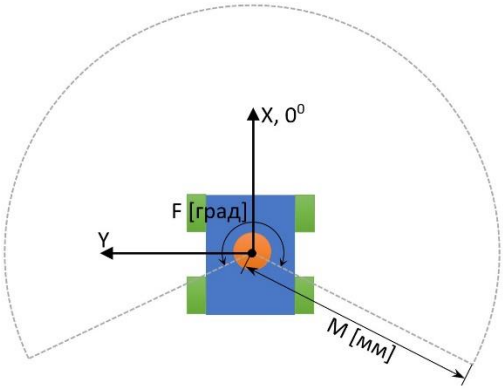
```
def find_most_common_color(path, name):
    # open the txt file with the image
    img = np.loadtxt(path)
    # convert to RGB
    img = rgb_to_hsv(img)
    # save image as txt file with width and height
    if name in should_not_check:
        return None

    print("")
    print(name)
    line = str(img.shape[0]) + ' ' + str(img.shape[1])

    img_data = img.reshape(-1, 3)
    with open('tests\\' + name, 'w') as f:
        for pixel in img_data:
            #convert pixel to hex
            hex_color = '%02x%02x%02x' % (pixel[0], pixel[1], pixel[2])
            f.write(hex_color + ' ')


    # np.savetxt('tests\\' + name, img_data, fmt='%d')
    with open('tests\\' + name, 'r+') as f:
        content = f.read()
        f.seek(0, 0)
        f.write(line.rstrip('\r\n') + '\n' + content)

    # reshape the image to be a list of pixels
    img = img.reshape((img.shape[0] * img.shape[1], 3))
    # cluster the pixel intensities
    clt = KMeans(n_clusters=3)
    clt.fit(img)
    # build a histogram of clusters and then create a figure
    # representing the number of pixels labeled to each color
    hist = centroid_histogram(clt)
    bar = plot_colors(hist, clt.cluster_centers_)
    # show our color bart in descending order
    plt.figure()
    plt.axis("off")
    plt.imshow(bar)
    # save image in the folder 'color_histograms'
    plt.savefig('color_histograms/' + name + '.png')
    # find the most common color
    most_common_color = clt.cluster_centers_[np.argmax(hist)]
    return most_common_color


# RGB to HSV without cv2
def rgb_to_hsv(color):
    r, g, b = list(map(int, color))
    print(r, g, b)
    r, g, b = r / 255.0, g / 255.0, b / 255.0
    cmax = max(r, g, b)
    cmin = min(r, g, b)
    diff = cmax - cmin
    if cmax == cmin:
        h = 0
    elif cmax == r:
        h = (60 * ((g - b) / diff) + 360) % 360
    elif cmax == g:
        h = (60 * ((b - r) / diff) + 120) % 360
    elif cmax == b:
        h = (60 * ((r - g) / diff) + 240) % 360
    if cmax == 0:
        s = 0
    else:
        s = (diff / cmax) * 256
    v = cmax * 256
    return list(map(int, [h, s, v]))


def color_to_answer(color):
    # RGB to HSV
    hsv = rgb_to_hsv(color)
```

```
    # get color description
    print(hsv)
    color_description = get_color_description(hsv)
    return color_description


check_colors = [((0, 20, 60, 255, 125, 255), 'RED'),
                ((21, 60, 60, 255, 125, 255), 'YELLOW'),
                ((61, 90, 60, 255, 125, 255), 'GREEN'),
                ((91, 120, 60, 255, 125, 255), 'GREEN'),
                ((121, 155, 60, 255, 125, 255), 'GREEN'),
                ((156, 180, 60, 255, 125, 255), 'BLUE'),
                ((181, 200, 60, 255, 125, 255), 'BLUE'),
                ((201, 260, 60, 255, 125, 255), 'BLUE'),
                ((261, 300, 60, 255, 125, 255), 'PINK'),
                ((301, 316, 60, 255, 125, 255), 'PINK'),
                ((317, 344, 60, 255, 125, 255), 'PINK'),
                ((345, 359, 60, 255, 125, 255), 'RED'),
                ((266, 338, 30, 255, 200, 255), 'PINK'),
                ((300, 359, 100, 160, 50, 200), 'RED'),
                ((266, 316, 20, 255, 80, 255), 'PINK'),
                ((156, 220, 20, 255, 200, 255), 'BLUE'),
                ((0, 359, 0, 125, 125, 255), 'WHITE'),
                ((0, 359, 0, 255, 0, 124), 'BLACK')]


def get_color_description(hsv):
    # H - Hue [0 - 359] hsv[0]
    # S - Saturation [0 - 255] hsv[1]
    # V - Value [0 - 255] hsv[2]
    for tmp_color in check_colors:
        if tmp_color[0][0] <= hsv[0] <= tmp_color[0][1] and tmp_color[0][2] <= hsv[1] <= tmp_color[0][3] and \
                tmp_color[0][4] <= hsv[2] <= tmp_color[0][5]:
            return tmp_color[1]
    return 'UNKNOWN'


should_not_check = []


def what_to_check(tests_names):
    for test in tests_names:
        test_name = test.split('.')[0]
        should_not_check.append(test_name)


if __name__ == '__main__':
    # create directories if it doesn't exist
    if not os.path.exists('color_histograms'):
        os.makedirs('color_histograms')
    # load all images from the folder 'task15\pictures'
    path = 'task15\\pictures'
    images = os.listdir(path)
    path2 = 'task15\\color_histograms'
    tests = os.listdir(path2)
    what_to_check(tests)
    for image in images:
        image_name = image.split('.')[0]
        image_path = os.path.join(path, image)
        try:
            answ = find_most_common_color(image_path, image_name)
            if answ is not None:
                answ = color_to_answer(answ)
                print(answ)
                with open(image_name + '.clue', 'w') as f:
                    f.write(answ)
        except:
            print('Error:' + image_name)
```

Вторая часть задач

Задача на оптимизацию работы лидара (2.1.1)

Продолжение задачи(1.1.3)



Изображение системы

Робот оборудован LIDAR'ом. Угол его сканирования F градусов, на эти F градусов приходится P измерений LIDAR'а. Максимальное расстояние измерения - до M миллиметров. Если LIDAR не видит объекта, то соответствующее измерение имеет максимальное значение. Нулевые координаты декартовой системы координат OXY совпадают с центром LIDAR'а, от которого он измеряет расстояние. Ось X этой системы координат совпадает с серединой сектора сканирования LIDAR'а. Кроме того, в системе присутствует полярная система координат. Ее полюс совпадает с началом декартовой системы координат и центром лидара. А полярная ось (нулевой луч) совпадает с осью X декартовой системы координат.

Контроллер робота не успевает обрабатывать все показания LIDAR'а с требуемым быстродействием. Гарантируется, что LIDAR видит один объект из следующего списка: параллелепипед / куб (тип 1), цилиндр (тип 2), прямая стена (тип 3), стена с выпуклым углом (тип 4), стена с вогнутым углом (тип 5). Гарантируется, что объект занимает не менее 1/10 от всего числа измерений (от числа P).

Формат входных данных

Первая строка содержит целое число — F — угол в градусах ($F \in [1; 360]$).

Вторая строка содержит целое число — P — количество измерений ($P \in [1; 4096]$).

Третья строка содержит целое число — M — дистанция измерения LIDAR'а в миллиметрах ($M \in [1; 4000]$).

Четвертая строка содержит P целых чисел - показания LIDAR'а.

Формат выходных данных

Целое число - тип объекта, который зафиксировал LIDAR.

Комментарии

Готовые библиотеки для решения задачи использовать запрещено. Примеры входных данных представлены по ссылке и не выводятся в самом задании.

Математическая система совпадает с задачами 1.1.2 и 1.1.3

Примеры

Примеры тестовых полигонов представлены по данной ссылке.

Решение

Псевдокод решения:

Read the angle F in degrees.
Read the number of dimensions P.
Read the lidar measurement distance M in millimeters.
Read the P lidar readings and store them in a list called m.
Calculate the angle between each lidar measurement:

If P is odd, divide F by 2 and then divide it by (P // 2). Store the result in delta_angle.
If P is even, divide F by (P - 1) and store the result in delta_angle.
Set the scanning angle s to F divided by 2.

Create an empty list called result to store intermediate results.

Create an empty list called temp to store transformed lidar readings.

Set x1 and y1 to positive infinity to initialize the coordinates of the first lidar measurement.

Create empty lists called xs1, ys1, xs2, ys2 to store intermediate coordinates.

Iterate from 0 to P-1:
a. Check if the lidar reading m[i] is not equal to M:

Create a 2D point g with the coordinates [m[i], 0].
Convert the scanning angle s to radians and store it in sf.
Create a rotation matrix using sf and store it in a variable called rotation_matrix.
Apply the rotation matrix to the point g, and store the result in a 2D point called x.
Invert the sign of the y-coordinate of x.
Append x to the temp list.
b. Subtract delta_angle from the scanning angle s.
Assign the transformed lidar readings from the temp list to the coords variable.

Set x1 and y1 to the x and y coordinates of the first point in coords.

Set x2 and y2 to the x and y coordinates of the last point in coords.

Create an empty set called m_values to store the rounded distances between points.

Iterate over the coordinates (x, y) in coords[1:-1]:

Calculate the distance d as the sum of the Euclidean distances between (x1, y1) and (x, y) and between (x2, y2) and (x, y).
Round the distance d to the nearest integer and add it to the m_values set.
If the size of the m_values set is less than or equal to 3:

Print 3 (indicating a straight wall type) and exit.
Set xc1 and yc1 to the coordinates of the middle point in coords.

Set xc2 and yc2 to the coordinates of the center between (x1, y1) and (x2, y2).

Set x3 and y3 to 0, 0 as the initial coordinates.

Set ds to negative infinity to store the maximum distance.

Iterate over the coordinates (x, y) in coords[1:-1]:

Calculate the distance d as the sum of the Euclidean distances between (x1, y1) and (x, y) and between (x2, y2) and (x, y).
If the distance d is greater than ds, update x3 and y3 with the current (x, y) coordinates, and update ds with d.
Set p1 to [x1, y1].

Set p2 to [x2, y2].

Set p3 to [x3, y3].

Calculate the angle between p1-p3 and p2-p3 using the calculate_angle function.

Calculate the Euclidean distances am and bm between p1-p2 and p4-p3, respectively.
Calculate the cosine of the angle using the dot product of the vectors a and b divided by (am * bm).
Use the inverse cosine (acos) function to calculate the angle in radians.
Convert the angle to degrees and round it to the nearest integer.
Assign the angle to a variable called angle.
If the angle is equal to 90 and the distance from (0, 0) to (x3, y3) is less than the distance from (0, 0) to (xc2, yc2):

Print 1 (indicating a parallelepiped/cube type) and exit.
If the distance from (0, 0) to (x3, y3) is greater than the distance from (0, 0) to (xc2, yc2):

Print 5 (indicating a wall with a concave angle type) and exit.
Create an empty set called m_values to store the rounded distances between points.

Iterate over the coordinates (x, y) in coords[1:coords.index([x3, y3])]:

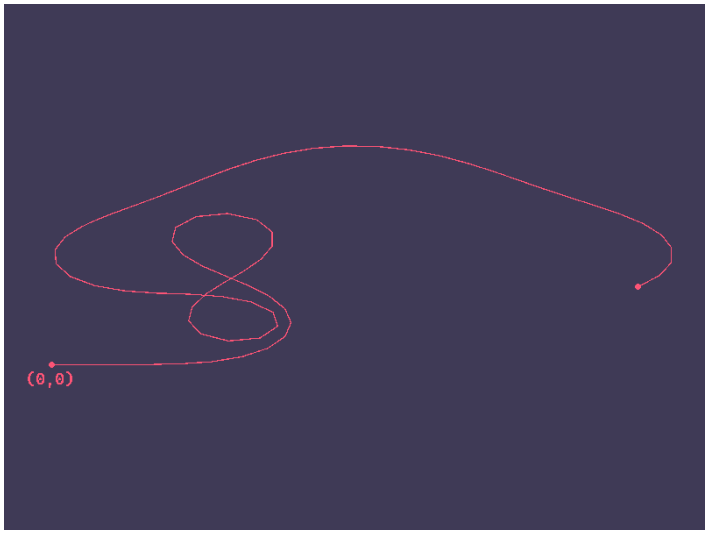
Calculate the distance d as the sum of the Euclidean distances between (x1, y1) and (x, y) and between (x3, y3) and (x, y).
Round the distance d to the nearest integer and add it to the m_values set.
If the size of the m_values set is less than or equal to 2:

Print 4 (indicating a wall with a convex angle type).
Otherwise:

Print 2 (indicating a cylinder type).

Задача на регулировку автоматизированного поезда (2.1.2)

Автоматизированный поезд с ПИД-контроллером движется вдоль заданного пути



Пример движения поезда

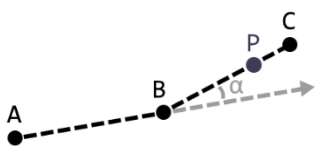
Известен путь движения поезда в виде N последовательных точек, расстояние между которыми равно $\Delta x = const$ = 1 м. Изначальная скорость поезда равна V, масса поезда равна M. Движение между соседними точками можно считать линейно-направленным и равноускоренным. Ускорение поезда происходит контролируемо с коллинеарной направлению движения силой $\vec{F}$ (0 <= $|\vec{F}|$ <= $FmaxF_{max}$).

Для упрощения считается, что других сил на поезд не оказывается, и изменение направления движения поезда не влияет на модуль его скорости (трение отсутствует). Изменение $\vec{F}$ применяется ПИД-контроллером мгновенно. При движении поезд сходит с рельс, если его модуль скорости $|\vec{V}|$ превышает максимальную скорость для текущего участка V_{max0}, задаваемую следующим образом:

$$V_{max0} = V_{max} * 2^{-(1-cos\alpha) * V_{factor}}$$

$$V_{factor} = const = 500$$

Где $Vmax$ - максимальная скорость для прямого участка пути (α = 0). Угол α является углом между текущим ($\vec{BC}$) и предыдущим ($\vec{AB}$) направлениями движения поезда (P) на сегментах пути:



Угол между сегментами

Движение поезда происходит с автоматизированным регулированием стандартного ПИД-контроллера, коэффициенты C_P, C_I, C_D которого известны. Ошибка контроллера $E0E_0$ в момент времени равна:

$$E_0 = 43 * V_{max0} - V_0$$

Где V_0 - текущая скорость поезда. E_0 напрямую численно передается для регулирования $\vec{F}$ в ПИД-контроллер:

$$C_{out} = C_P * e(t) + C_I * \int e(t) * dt + C_D * \frac{d}{dt} e(t)$$

ПИД-контроллер производит все обновления с частотой $FREQ = const$ = 1000Гц.

Формат входных данных

Первая строка содержит 1 целое число и 4 вещественных числа соответственно через пробел: N, V, F_{max}, M, P_{max}, где:

• N — количество точек пути поезда ($0 < N < 10^4$);

• V — начальная скорость поезда в м/с ($0 < V < 10^2$);

• F_{max}— модуль максимально применимой ПИД-контроллером силой в ньютонах ($1 < F_{max} < 10^3$);

• M — масса поезда в килограммах ($1 < M < 10^3$);

• P_{max}– максимальный импульс поезда ($P_{max} = M * V_{max}$) для движения по прямой (α = 0) в кг*м/с ($1 < P_{max} < 10^4$).

Вторая строка содержит 3 вещественных числа соответственно через пробел: C_P, C_I, C_D, где:

• C_P — пропорциональная составляющая ПИД-контролера ($0 <= C_P < 10^2$);

• C_I– интегрирующая составляющая ПИД-контролера ($0 <= C_I < 10^2$);

• C_D– дифференцирующая составляющая ПИД-контролера ($0 <= C_D < 10^2$);

Далее идут N строк, состоящие из 2 вещественных числа через пробел: X, Y :

• X — координата пути по оси X в метрах ($-10^6 < X < 10^6$);

• Y — координата пути по оси Y в метрах ($-10^6 < Y < 10^6$).

Формат выходных данных

Первая строка — единственное слово $DONE$, если симуляция произведена успешно. Слово $FAIL$, если происходит нарушение указанных требований к скорости поезда;

Вторая строка — одно вещественное число с точностью до сотых — время T в секундах в рамках допустимой ошибки, за которое поезд пройдет указанный или доступный путь:

$$T_{correct} * 0.95 <= T <= T_{correct} * 1.05$$

Комментарии

Готовые библиотеки для решения задачи использовать запрещено. Примеры входных данных представлены по ссылке и не выводятся в самом задании.

Примеры

Примеры тестовых путей движения поезда представлены по данной ссылке.

Предоставленные тесты делятся на пять категорий по мере повышения сложности:

• Для тестов [1; 10] гарантируется, что $V \gg 0$, C_P, C_I, $C_D = 0$, $\forall \alpha (\alpha = const = 0)$.

• Для тестов [11; 30] гарантируется, что $V \gg 0$, C_P, C_I, $C_D = 0$.

• Для тестов [31; 40] гарантируется, что C_I, $C_D = 0$.

• Для тестов [41; 50] гарантируется, что $C_I = 0$.

• Для тестов [51; 100] дополнительных условий не предусмотрено.

Решение

Код решения на языке Python:

```
import math
import os

def distance(x0, y0, x, y):
    dx = (x - x0)
    dy = (y - y0)
    return dx * dx + dy * dy

def calc_angle_cos(x0, y0, x1, y1, x2, y2):
    v01x = x1 - x0
    v01y = y1 - y0
    v12x = x2 - x1
    v12y = y2 - y1
    # Dot product (v01 v12)
    dp = (v01x * v12x) + (v01y * v12y)
    # v01 len
    v01l = math.sqrt(v01x * v01x + v01y * v01y)
    # v12 len
    v12l = math.sqrt(v12x * v12x + v12y * v12y)
    # Angle cos
    a_cos = dp / v01l / v12l
    return a_cos

def calc_max_velocity(angle_cos, VMax, VFactor):
    result = VMax * math.pow(2, -(1 - angle_cos) * VFactor)
    return result

TEST_COUNT = 100

#KP = 0.5
#KI = 10.0
```

```
#KD = 0.3

VCOEF = 500.0

for index in range(TEST_COUNT):
    print('Solving Test', str(index) + '/' + str(TEST_COUNT - 1) + '...')

    # Simulation delta time in seconds
    STEP_DT = 0.001

    f_input = open(os.path.join(os.path.dirname(__file__), "../tests/" + str(index)), 'r')
    lines = f_input.readlines()
    args = lines[0].split(' ')
    N = int(args[0])
    V = float(args[1])
    FMax = float(args[2])
    M = float(args[3])
    MVMax = float(args[4])
    VMax = MVMax / M

    args = lines[1].split(' ')
    CP = float(args[0])
    CI = float(args[1])
    CD = float(args[2])

    # Max acceleration
    AMax = FMax / M

    # Train Velocity
    tr_v = V
    # Current Sample
    cur_s = 0
    # Current per-sample distance
    cur_sd = 0.0
    # Controller integral
    c_i = 0.0
    # Current time (answer)
    t = 0.0
    # Previous sample pos
    x0 = 0.0
    y0 = 0.0
    # Previous error
    prev_e = 0.0
    # Fail flag
    fl_fail = False

    debug_d = 0.0
    debug_max_alpha = 0.0
    debug_min_alpha = math.inf
    debug_min_deltav = math.inf

    while cur_s < N:
        # Target sample pos
        s_pos = lines[cur_s + 2].split(' ')
        x2 = float(s_pos[0])
        y2 = float(s_pos[1])

        # Current sample pos
        x1 = 0.0
        y1 = 0.0
        if cur_s > 0:
            ps_pos = lines[cur_s + 1].split(' ')
            x1 = float(ps_pos[0])
            y1 = float(ps_pos[1])

        # Distance between current and previous samples
        s_d = distance(x1, y1, x2, y2)

        s_maxvelocity = VMax
        s_angle_cos = 0.0
        if cur_s > 1:
            s_angle_cos = calc_angle_cos(x0, y0, x1, y1, x2, y2)
            s_maxvelocity = calc_max_velocity(s_angle_cos, VMax, VCOEF)

        # Current error
        cur_error = s_maxvelocity * 3.0 / 4.0 - tr_v
```

```
    c_i += cur_error * STEP_DT

    # Current acceleration
    a = 0.0 # 0 <-> FMax / M
    f = max(min(CP * cur_error + CI * c_i + CD * (cur_error - prev_e) / STEP_DT, FMax), -FMax)
    a = f / M

    debug_min_deltav = min(debug_min_deltav, s_maxvelocity - tr_v)
    #print('V:', round(tr_v, 3), '/', round(s_maxvelocity, 3), 'ERR:', round(cur_error, 3), 'ACC:', round(a, 3))

    if tr_v >= s_maxvelocity:
        #print('SIMULATION FAIL', str(tr_v), str(s_maxvelocity))
        fl_fail = True
        break

    # Delta distance per this step
    d = (STEP_DT * tr_v) + (a * STEP_DT * STEP_DT) / 2.0

    if cur_sd > s_d:
        print("We have a problem!") #ToDo
        break

    cur_sd += d
    tr_v += a * STEP_DT

    # If we go over sample distance
    if cur_sd >= s_d:
        cur_s += 1
        cur_sd -= s_d
        debug_d += s_d

        if cur_s > 0:
            x0 = x1
            y0 = y1

    t += STEP_DT
    prev_e = cur_error


f_clue = open(os.path.join(os.path.dirname(__file__), "../tests/" + str(index)) + '.clue', 'w')
if fl_fail:
    f_clue.write('FAIL\n')
    print('Done! Fail', round(t, 2))
else:
    f_clue.write('DONE\n')
    print('Done! Success', round(t, 2))
f_clue.write(str(round(t, 2)))
f_clue.close()

#print('MIN DELTA VELOCITY: ', debug_min_deltav)
```