Беспилотные авиационные системы

Второй отборочный этап

Задача IV.1. Обнаружение объекта системой технического зрения (35 баллов)

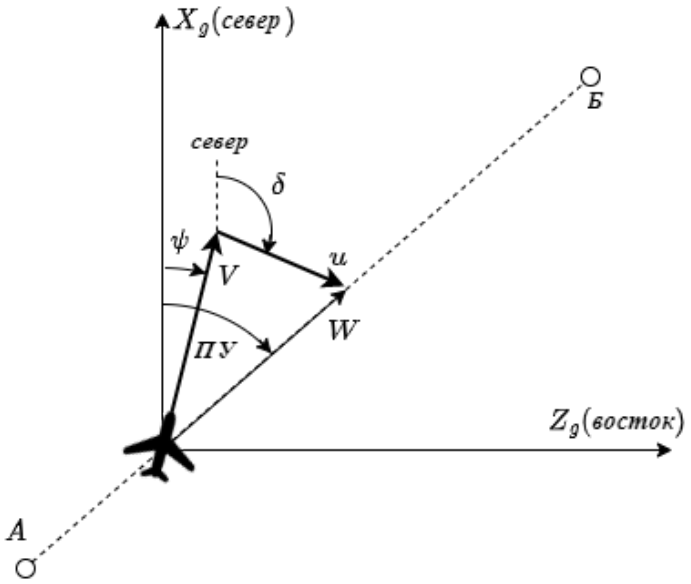
Темы: математика, программирование.

Поиск объектов при помощи системы технического зрения является неотъемлемой частью задачи финала.

Условие

При решении задач поиска объектов системой технического зрения БЛА требуется не только обрабатывать полученные изображения и распознавать на них искомые объекты, но и учитывать текущие параметры полёта БЛА для точного определения местоположения объекта относительно самолёта. При этом бортовые системы навигации современных БЛА, как правило, содержат большое количество датчиков, обеспечивающих избыточность измерений. Это позволяет получать точное навигационное решение даже в случае отказа одного из датчиков.

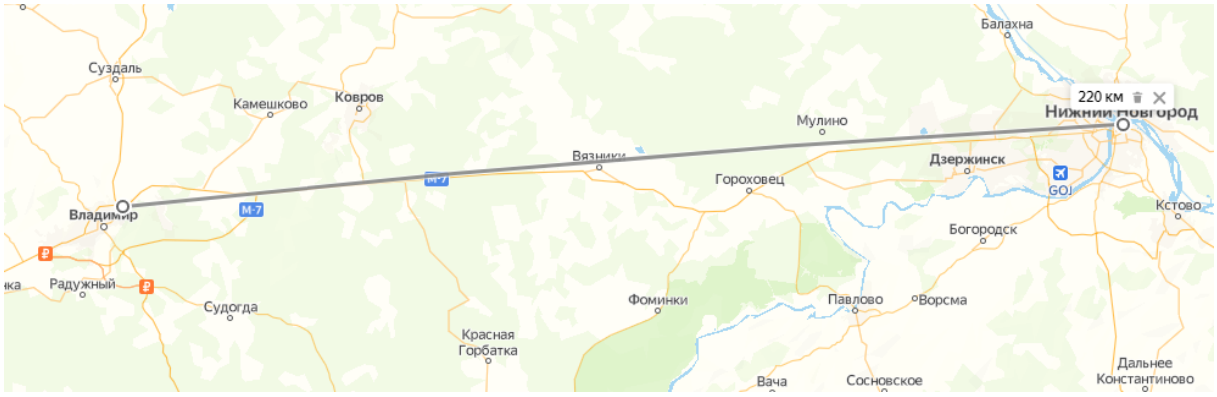
В навигации широко используется понятие навигационного треугольника скоростей. В упрощённом виде этот треугольник для случая полёта из точки A в точку $Б$ представлен на рисунке.



Здесь обозначены: V — истинная воздушная скорость полёта БЛА (скорость движения относительно воздуха); u — вектор скорости ветра; δ — угол ветра (навигационного направления ветра); ψ — угол курса (угол между направлением на север и продольной осью БЛА); W — путевая скорость (скорость относительно земли); $ПУ$ — путевой угол (угол между направлением на север и вектором путевой скорости).

В общем случае расчёт этого треугольника сводится к определению всех указанных параметров. Это позволяет определить взаимосвязь между собственным движением БЛА относительно воздуха и его перемещением относительно земли.

Предположим, что БЛА совершает перелёт из Владимира в Нижний Новгород по прямой линии (координаты точки старта: $\phi_{\text{с}} = 56,169261$, $\lambda_{\text{с}} = 40,472165$ (град.), координаты точки финиша: $\phi_{\text{ф}} = 56,169261$, $\lambda_{\text{ф}} = 40,472165$ (град.).



В процессе полёта БЛА осуществляет непрерывную фотосъёмку подстилающей поверхности на предмет наличия ориентиров для корректировки своего местоположения. Примерно в середине полёта происходит отказ бортового измерителя угла курса БЛА, но благодаря корректировке навигационного решения по системе технического зрения, он может и дальше оставаться на линии заданного пути (на прямой линии между точками старта и финиша).

Для этого система технического зрения должна корректно определять смещение ориентира в кадре камеры относительно текущего местоположения БЛА вдоль осей X_g и Z_g.

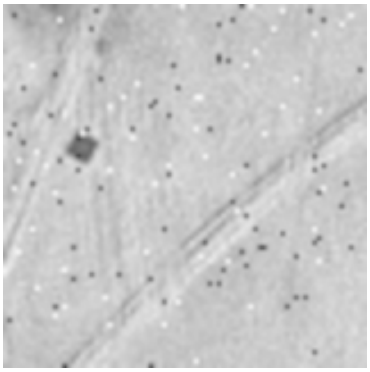
Напишите программу на языке C++ или Python, определяющую смещение ориентира в кадре камеры относительно БЛА (в м) по заданным параметрам: истинная воздушная скорость $V = 16,58$ м/с, скорость ветра $u = 3,46$ м/с, угол ветра $\delta = 137,2$ град, высота полёта $H = 50$ м, угол обзора камеры БЛА (горизонтальный и вертикальный) $FOV = 60$ град. Углы тангажа и крена БЛА равны нулю. Координаты (географические широта и долгота) точки старта: $\phi_{\text{с}} = 56,169261$, $\lambda_{\text{с}} = 40,472165$ (град.), координаты точки финиша: $\phi_{\text{ф}} = 56,169261$, $\lambda_{\text{ф}} = 40,472165$ (град.). Радиус Земли $R = 6366,2$ км.

Изображение подстилающей поверхности имеет следующий вид.

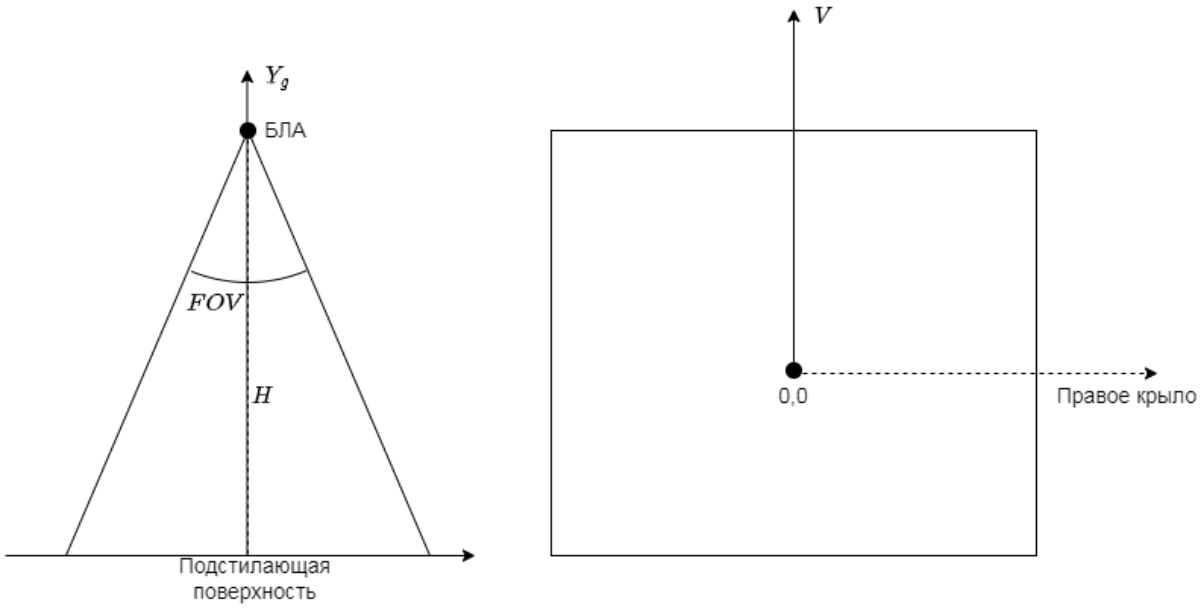
В силу технических ограничений бортовая камера позволяет получить только

чёрно-белые снимки низкого разрешения с наличием шумов (дефектов) изображения.

Исходным изображением в рамках задачи является чёрно-белое изображение в формате PGM. Размер изображения 80×80 пк. На изображении присутствует контрастный ориентир. Пример входного изображения показан на рисунке.



За текущее местоположение БЛА следует принимать центр снимка с координатами 0, 0 м. Истинным местоположением ориентира считается его геометрический центр. При расчётах считать, что подстилающая поверхность ровная и её высота равна нулю. Система координат снимка показана на рисунке.



Для нахождения точного ответа вам потребуется рассчитать угол курса БЛА, осуществить фильтрацию исходного изображения (метод фильтрации участники выбирают самостоятельно), рассчитать смещение объекта вдоль осей X_g и Z_g.

Эта задача имеет гибкий критерий оценки. Максимальное количество баллов, которые можно получить за решение — 35. Оценивается радиальное смещение $\Delta = \sqrt{\Delta x^2 + \Delta y^2}$ от истинного местоположения ориентира. За каждые 0,1 м ошибки списывается 1 балл (таким образом, если ошибка решения составляет 0,099 м и меньше, за решение задачи начисляется 35 баллов).

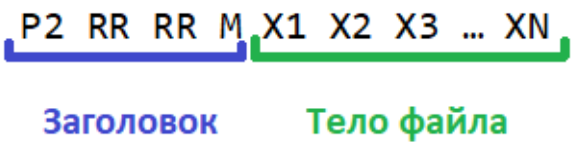
На платформе stepik вам доступен один открытый вариант задания и три закрытых. При отправке решения система автоматической проверки сообщит вам величину ошибки и ожидаемую оценку для каждого из четырёх вариантов.

Итоговое оценивание будет осуществляться путём усреднения результата по трём закрытым вариантам (считается среднее арифметическое ошибки).

Дополнительные варианты задания для тестирования ваших алгоритмов доступны в приложении.

Формат входных данных

На вход программы поступает текст PGM-файла, сохраненного в текстовом формате (на платформе stepik содержимое PGM-файла задаётся в виде строки через консоль). Текст файла состоит из двух основных частей — заголовка и тела. Описание структуры PGM-файла можно найти по ссылке в списке литературы. Все числа представляют собой целые значения, разделённые пробелами. Тело PGM-файла содержит 6400 значений, описывающих яркость каждого из пикселей изображения размером 80×80.



Формат выходных данных

Необходимо вывести на стандартный вывод смещение найденного ориентира относительно БЛА вдоль осей X_g и Z_g в метрах с точностью до 2 знаков после запятой.

Ссылка на набор тестов: https://disk.yandex.ru/d/4cNDSsudecT35A.

Решение

1. Осуществить фильтрация входного изображения (например, использовать медианный фильтр из библиотеки opencv).
2. Увеличить контрастность размытого изображения на 10%.
3. Найти все пиксели изображения, яркость которых не превышает 150, и записать в отдельные массивы.
4. Найти средние индексы всех пикселей, соответствующих ориентиру.
5. Рассчитать расстояние между точками A и $Б$ в м.
6. Определить путевой угол через тангенс катетов.
7. Определить угол курса через теорему синусов.
8. Определить ширину одного пикселя в метрах.
9. Рассчитать смещение ориентира относительно БЛА в связанной системе координат.
10. Перевести смещение в географическую нормальную систему координат.

Пример программы-решения

Ниже представлено решение на языке Python 3.

```
import numpy as np
import cv2
import math

fs = 56.169261
ls = 40.472165
ff = 56.331157
lf = 44.009462

fcalc = (fs + ff) / 2.0
lcalc = (ls + lf) / 2.0
dx = (ff - fs) * 111.111  # km on north
dz = (lf - ls) * 111.111 * math.cos(fcalc * np.pi / 180.0)

PU = math.atan(dz / dx)
u = 3.46
V = 16.58
sig = 137.2 * np.pi / 180.0
al = sig - PU
psi = PU - math.asin(math.sin(al) * u / V)

# Read console input
arr = [i for i in input().split()]
head = arr[0:4]
body = []
for i in range(4, len(arr)):
    body.append(float(arr[i]))

blank = np.zeros((80, 80, 1), dtype="uint8")
xc = []
zc = []
for i in range(0, len(body)):
    row = int(i / 80)
    col = i - row * 80
    blank[row][col] = body[i]

rows, cols, ch = blank.shape
im_filt = blank
im_filt = cv2.medianBlur(blank, 5)
for i in range(0, rows):
    for j in range(0, cols):
        im_filt[i][j] = int(im_filt[i][j] * 1.1)
        if (im_filt[i][j]) > 255:
            im_filt[i][j] = 255

for i in range(0, rows):
    for j in range(0, cols):
        if im_filt[i][j] < 150:
            xc.append(i)
            zc.append(j)
dx = 40.0 - sum(xc) / float(len(xc))
dz = sum(zc) / float(len(zc)) - 40.0

FOV = 60 * np.pi / 180.0
H = 50
px = math.tan(FOV / 2) * H / 40
dxm = dx * px
dzm = dz * px
dxmg = dxm * math.cos(psi) - dzm * math.sin(psi)
dzmg = dxm * math.sin(psi) + dzm * math.cos(psi)
```

```
61  print("{:.2f} {:.2f}".format(dxmg, dzmg))
```

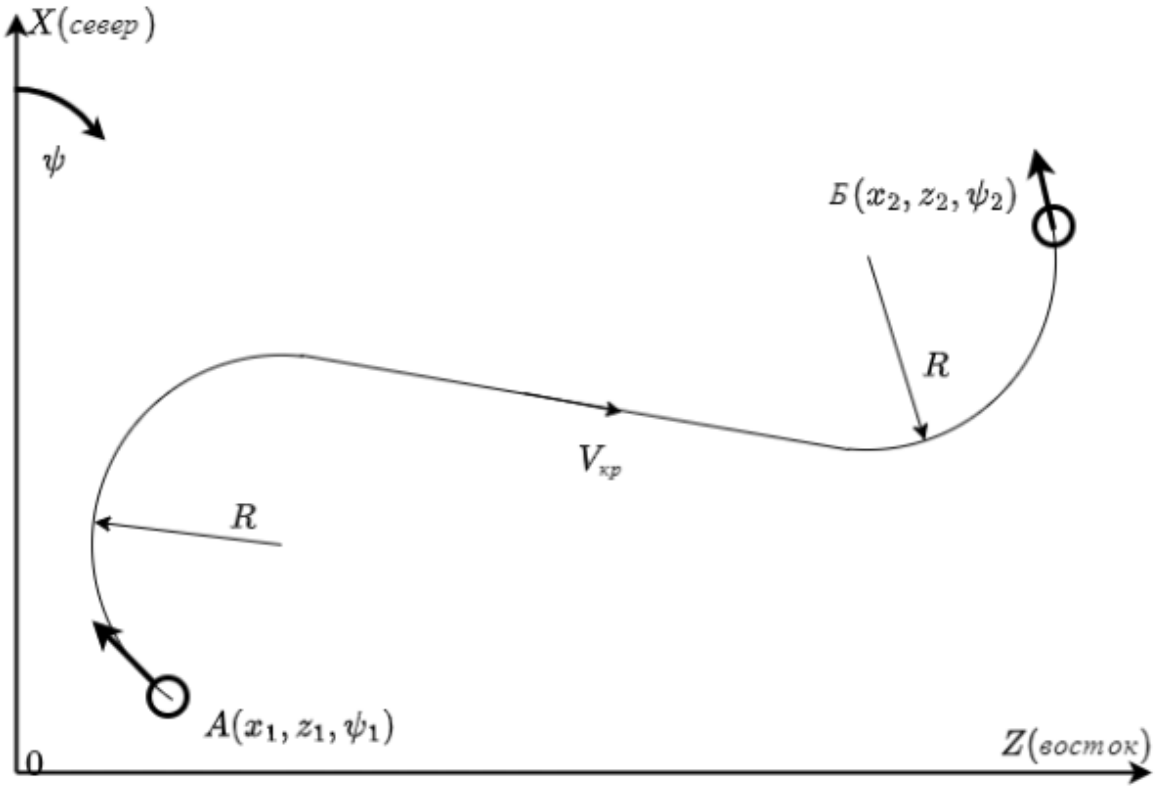
Задача IV.2. Оптимальная траектория полёта (35 баллов)

Темы: математика, физика, программирование, планирование пути.

В финале участникам предстоит решать задачу построения системы автоматического управления полётом БЛА. Эта задача предназначена для ознакомления с основными принципами построения траекторий полёта.

Условие

При построении системы автоматического управления (САУ) БЛА одной из ключевых задач является оптимизация траектории его полёта. Критерии оптимальности могут быть разными: затраченное время, пройденный путь, количество разворотов, энергопотребление и т. д. Однако одним из самых распространённых критериев эффективности является минимизация времени полёта.



Исследования показывают, что, при условии достаточной удалённости двух точек друг от друга, кратчайшая траектория полёта из точки A (с начальным углом курса) в точку $Б$ (с заданной ориентацией по углу курса) будет состоять из разворота по дуге окружности, полёта по прямой и ещё одного разворота.

Для достижения критерия минимизации времени полёта БЛА должен двигаться по кратчайшей траектории с максимально возможной скоростью.

Допустим, что БЛА совершает полёт на постоянной высоте и может осуществлять разворот только по окружности заданного радиуса $R = 100$ м. При этом максимальное значение угла крена составляет $\gamma = 45$ град.

Связь мгновенного радиуса разворота с углом крена описывается соотношением:

$$R = \frac{V^2}{g \cdot \tan\gamma}.$$

Напишите программу на языке C++ или Python, осуществляющую расчёт минимально возможной длины траектории полёта из точки A в точку $Б$ с учётом ограничений на углы курса в начальной и конечной точках. При этом ускорение свободного падения $g = 9,81$ м/с^2, максимальная угловая скорость разворота по крену $\dot{\gamma}_{max} = 0,2618$ 1/с, крейсерская скорость полёта БЛА по прямой $V_{\text{кр}} = 15$ м/с, а максимальное линейное ускорение не ограничено.

Гарантируется, что расстояние между точками A и $Б$ достаточно велико, чтобы оптимальная траектория включала в себя участок прямолинейного полёта.

Примечание: в начальный момент времени угол крена БЛА всегда равен нулю. И в конце траектории угол также должен быть нулевым (осуществляется выход из виража).

Эта задача имеет гибкий критерий оценки. Максимальное количество баллов, которые можно получить за решение — 35. За каждые 0,1 м ошибки определения длины кратчайшей траектории полёта будет сниматься один балл (таким образом, если ошибка расчёта длины составляет 0,09 м — за решение задачи начисляется 35 баллов).

На платформе stepik вам доступно 4 открытых варианта задания и 2 закрытых. При отправке решения система автоматической проверки сообщит вам величины ошибок и ожидаемую оценку для каждого из вариантов.

Итоговое оценивание будет осуществляться путём усреднения результата по двум закрытым вариантам.

Формат входных данных

Входными данными являются параметры начальной и конечной точек маршрута в формате:

```
XX ZZ HH
XX ZZ HH
```

Здесь `XX` — координата вдоль оси X (м), `ZZ` — координата вдоль оси Z (м), `HH` — угол курса (град.) в пределах от -180 до 180.

Формат выходных данных

Необходимо вывести на стандартный вывод длину оптимальной траектории между двумя точками (в м). Выходное значение выводится с точностью до 2 знаков после запятой.

Примеры

Пример №1

Стандартный ввод
0 0 0 0 500 90
Стандартный вывод
569.93

Ссылка на набор тестов: https://disk.yandex.ru/d/w4cpvbYEK9o5qQ.

Решение

1. Определить четыре возможных конфигурации траектории между точками.
2. Выбрать наикратчайшую траекторию.
3. Рассчитать время перемещения БЛА по каждой из траекторий.
4. Определить наименьшее время.

Пример программы-решения

Ниже представлено решение на языке Python 3.

```
import math

arr1 = [float(i) for i in input().split()]
arr2 = [float(i) for i in input().split()]
x1 = arr1[0]
z1 = arr1[1]
h1 = arr1[2]

x2 = arr2[0]
z2 = arr2[1]
h2 = arr2[2]

class Position:
    x = 0.0
    z = 0.0
    h = 0.0

    def __init__(self, xx, zz, hh):
        self.x = xx
        self.z = zz
        self.h = hh

class Trajectory:
    pos_circ_1 = Position(0.0, 0.0, 0.0)
    pos_circ_2 = Position(0.0, 0.0, 0.0)
    pos_switch_1 = Position(0.0, 0.0, 0.0)
    pos_switch_2 = Position(0.0, 0.0, 0.0)
    alpha = 0.0
    beta = 0.0
    lenseg = [0.0, 0.0, 0.0]
    lensum = 0.0
    def __init__(self):
```

```
33          self.lenseg = [0.0, 0.0, 0.0]
34
35  def ShiftPosition(pos, alpha, r):
36      return Position(pos.x + r * math.cos(alpha), pos.z + r * math.sin(alpha),
        ↪  pos.h)
37
38  def PositionDifference(pos1, pos2):
39      return Position(pos2.x - pos1.x, pos2.z - pos1.z, 0.0)
40
41  def LimitAngle(angle):
42      # limit to 2*pi
43      angle = math.fmod(angle, math.pi*2)
44      # limit to positive
45      while angle < 0:
46          angle = angle + math.pi * 2
47      return angle
48
49  def Construct_RSR(pos1, pos2, r):
50      traj = Trajectory()
51      # circle centers
52      traj.pos_circ_1 = ShiftPosition(pos1, pos1.h + math.pi/2, r)
53      traj.pos_circ_2 = ShiftPosition(pos2, pos2.h + math.pi/2, r)
54      # relative distance between the circle centers
55      d_pos_circ = PositionDifference(traj.pos_circ_1, traj.pos_circ_2)
56      d = math.sqrt(d_pos_circ.x**2 + d_pos_circ.z**2)
57      theta = math.atan2(d_pos_circ.z, d_pos_circ.x)
58      # switching points
59      traj.pos_switch_1 = ShiftPosition(traj.pos_circ_1, theta - math.pi/2, r)
60      traj.pos_switch_2 = ShiftPosition(traj.pos_circ_2, theta - math.pi/2, r)
61      # turning angles
62      traj.alpha = theta - pos1.h
63      traj.beta = pos2.h - theta
64      # adjust angles
65      traj.alpha = LimitAngle(traj.alpha)
66      traj.beta = LimitAngle(traj.beta)
67      # calculate path lenghts
68      traj.lenseg[0] = traj.alpha * r
69      traj.lenseg[1] = d
70      traj.lenseg[2] = traj.beta * r
71      d_pos = PositionDifference(traj.pos_switch_1, traj.pos_switch_2)
72      d1 = math.sqrt(d_pos.x**2 + d_pos.z**2)
73      traj.lenseg[1] = d1
74      #print(traj.lenseg)
75      traj.lensum = sum(traj.lenseg)
76      return traj
77
78  def Construct_LSL(pos1, pos2, r):
79      traj = Trajectory()
80      # circle centers
81      traj.pos_circ_1 = ShiftPosition(pos1, pos1.h - math.pi/2, r)
82      traj.pos_circ_2 = ShiftPosition(pos2, pos2.h - math.pi/2, r)
83      # relative distance between the circle centers
84      d_pos_circ = PositionDifference(traj.pos_circ_1, traj.pos_circ_2)
85      d = math.sqrt(d_pos_circ.x**2 + d_pos_circ.z**2)
86      theta = math.atan2(d_pos_circ.z, d_pos_circ.x)
87      # switching points
88      traj.pos_switch_1 = ShiftPosition(traj.pos_circ_1, theta + math.pi/2, r)
89      traj.pos_switch_2 = ShiftPosition(traj.pos_circ_2, theta + math.pi/2, r)
90      # turning angles
91      traj.alpha = pos1.h - theta
```

```
92      traj.beta = theta - pos2.h
93      # adjust angles
94      traj.alpha = LimitAngle(traj.alpha)
95      traj.beta = LimitAngle(traj.beta)
96      # calculate path lenghts
97      traj.lenseg[0] = traj.alpha * r
98      traj.lenseg[1] = d
99      traj.lenseg[2] = traj.beta * r
100     d_pos = PositionDifference(traj.pos_switch_1, traj.pos_switch_2)
101     d1 = math.sqrt(d_pos.x**2 + d_pos.z**2)
102     traj.lenseg[1] = d1
103     #print(traj.lenseg)
104     traj.lensum = sum(traj.lenseg)
105     return traj
106
107 def Construct_RSL(pos1, pos2, r):
108     traj = Trajectory()
109     # circle centers
110     traj.pos_circ_1 = ShiftPosition(pos1, pos1.h + math.pi/2, r)
111     traj.pos_circ_2 = ShiftPosition(pos2, pos2.h - math.pi/2, r)
112     # relative distance between the circle centers
113     d_pos_circ = PositionDifference(traj.pos_circ_1, traj.pos_circ_2)
114     d = math.sqrt(d_pos_circ.x**2 + d_pos_circ.z**2)
115     eta = math.atan2(d_pos_circ.z, d_pos_circ.x)
116     gamma = math.acos(2 * r / d)
117     L = math.sqrt(d*d - 4.0*r*r)
118     theta = eta - gamma + math.pi / 2
119     # switching points
120     traj.pos_switch_1 = ShiftPosition(traj.pos_circ_1, theta - math.pi/2, r)
121     traj.pos_switch_2 = ShiftPosition(traj.pos_circ_2, theta + math.pi/2, r)
122     # turning angles
123     traj.alpha = theta - pos1.h
124     traj.beta = theta - pos2.h
125     # adjust angles
126     traj.alpha = LimitAngle(traj.alpha)
127     traj.beta = LimitAngle(traj.beta)
128     # calculate path lenghts
129     traj.lenseg[0] = traj.alpha * r
130     traj.lenseg[1] = d
131     traj.lenseg[2] = traj.beta * r
132     d_pos = PositionDifference(traj.pos_switch_1, traj.pos_switch_2)
133     d1 = math.sqrt(d_pos.x**2 + d_pos.z**2)
134     traj.lenseg[1] = d1
135     #print(traj.lenseg)
136     traj.lensum = sum(traj.lenseg)
137     return traj
138
139 def Construct_LSR(pos1, pos2, r):
140     traj = Trajectory()
141     # circle centers
142     traj.pos_circ_1 = ShiftPosition(pos1, pos1.h - math.pi/2, r)
143     traj.pos_circ_2 = ShiftPosition(pos2, pos2.h + math.pi/2, r)
144     # relative distance between the circle centers
145     d_pos_circ = PositionDifference(traj.pos_circ_1, traj.pos_circ_2)
146     d = math.sqrt(d_pos_circ.x**2 + d_pos_circ.z**2)
147     eta = math.atan2(d_pos_circ.z, d_pos_circ.x)
148     gamma = math.acos(2 * r / d)
149     L = math.sqrt(d*d - 4.0*r*r)
150     theta = eta + gamma - math.pi / 2
151     # switching points
```

```
152     traj.pos_switch_1 = ShiftPosition(traj.pos_circ_1, theta + math.pi/2, r)
153     traj.pos_switch_2 = ShiftPosition(traj.pos_circ_2, theta - math.pi/2, r)
154     # turning angles
155     traj.alpha = pos1.h - theta
156     traj.beta = pos2.h - theta
157     # adjust angles
158     traj.alpha = LimitAngle(traj.alpha)
159     traj.beta = LimitAngle(traj.beta)
160     # calculate path lenghts
161     traj.lenseg[0] = traj.alpha * r
162     traj.lenseg[1] = d
163     traj.lenseg[2] = traj.beta * r
164     #print(traj.lenseg)
165     d_pos = PositionDifference(traj.pos_switch_1, traj.pos_switch_2)
166     d1 = math.sqrt(d_pos.x**2 + d_pos.z**2)
167     traj.lenseg[1] = d1
168     traj.lensum = sum(traj.lenseg)
169     return traj
170 
171 def AccelerationToMax(r, rate, dt):
172     gamma_max = 45 * math.pi / 180.0
173     g = 9.81
174     gamma = 0
175     path_traveled = 0
176     time_spent = 0
177     while gamma < gamma_max:
178         gamma += rate * dt
179         time_spent += dt
180         path_traveled += math.sqrt(g * r * math.tan(gamma)) * dt
181     while gamma >= rate * dt:
182         gamma -= rate * dt
183         time_spent += dt
184         path_traveled += math.sqrt(g * r * math.tan(gamma)) * dt
185     return (time_spent, path_traveled)
186 
187 def TimeToRotate(arclen, r, rate, dt):
188     g = 9.81
189     path_traveled = 0
190     time_spent = 0
191     gamma = 0.0
192     #print("Estimating for small arc: {}".format(arclen))
193     path2travel = arclen / 2
194     while path2travel > 0:
195         gamma += rate * dt
196         time_spent += dt
197         step = math.sqrt(g * r * math.tan(gamma)) * dt
198         path_traveled += step
199         path2travel -= step
200     path2travel = arclen / 2
201     while path2travel > 0 and gamma > 0:
202         gamma -= rate * dt
203         time_spent += dt
204         step = math.sqrt(g * r * math.tan(gamma)) * dt
205         path_traveled += step
206         path2travel -= step
207     return (time_spent, path_traveled)
208 
209 def EstimateTime(tra, r):
210     #print(tra.lenseg)
211     #print(tra.lensum)
```

```
212     Vcr = 15.0  # cruise velocity
213     g = 9.81
214     gamma_max = 45 * math.pi / 180.0
215     time_straight = tra.lenseg[1] / Vcr
216     V_max = math.sqrt(g * r * math.tan(gamma_max))
217     V_1 = math.sqrt(g * r * math.tan(15 * math.pi / 180.0))
218     V_2 = math.sqrt(g * r * math.tan(30 * math.pi / 180.0))
219     #print("Vmax: {}".format(V_max))
220     accelerating_len = V_1 + V_2 + V_max
221     #print("accelerating_len: {}".format(accelerating_len))
222     time_max, path_max = AccelerationToMax(r, 15.0 * math.pi / 180.0, 0.1)
223     #print("time_max: {} path_max: {}".format(time_max, path_max))
224     time_turn_1 = 0
225     if (tra.lenseg[0] >= path_max):
226         max_vel_seg = tra.lenseg[0] - path_max
227         time_turn_1 += max_vel_seg / V_max  # maximum velocity time (gamma = 45)
228         time_turn_1 += time_max   # acceleration and deacceleration time (gamma 0
        ↪  -> 45) (gamma 45 -> 0)
229         path_traveled = tra.lenseg[0]
230     else:
231         time_turn_1, path_traveled = TimeToRotate(tra.lenseg[0], r, 45.0 * math.pi
        ↪  / 180.0, 0.1)
232     #print("time_turn_1: {}".format(time_turn_1))
233     #print("path_traveled: {} traj.lenseg[0]: {}".format(path_traveled,
    ↪  traj.lenseg[0]))
234     time_turn_2 = 0
235     if (tra.lenseg[2] >= 2 * path_max):
236         max_vel_seg = tra.lenseg[2] - path_max
237         time_turn_1 += max_vel_seg / V_max  # maximum velocity time (gamma = 45)
238         time_turn_1 += time_max   # acceleration and deacceleration time (gamma 0
        ↪  -> 45) (gamma 45 -> 0)
239         path_traveled = tra.lenseg[2]
240     else:
241         time_turn_2, path_traveled = TimeToRotate(tra.lenseg[2], r, 45.0 * math.pi
        ↪  / 180.0, 0.1)
242     #print("time_turn_2: {}".format(time_turn_2))
243     #print("path_traveled: {} traj.lenseg[2]: {}".format(path_traveled,
    ↪  traj.lenseg[2]))
244     return time_straight + time_turn_1 + time_turn_2
245
246 R = 100  # meters
247 start = Position(x1, z1, h1 * math.pi / 180.0)
248 goal = Position(x2, z2, h2 * math.pi / 180.0)
249 tr = []
250 tr.append(Construct_RSR(start, goal, R))
251 tr.append(Construct_LSL(start, goal, R))
252 tr.append(Construct_RSL(start, goal, R))
253 tr.append(Construct_LSR(start, goal, R))
254 #print("")
255
256 inx = 0
257 cost = 1e10
258 spent = 1e10
259 for i in range(0, 4):
260     sp = EstimateTime(tr[i], R)
261     #print("total time spent: {}".format(sp))
262     if (tr[i].lensum < cost):
263         cost = tr[i].lensum
264         inx = i
265     if (sp < spent):
```

```
            spent = sp

    #print(inx)
    print("{:.2f} {:.2f}".format(cost, spent))
    #print("")
```

Задача IV.3. Разработка канала управления курсом БЛА (30 баллов)

Темы: программирование, физика, управление.

В финале участникам предстоит решать задачу построения системы автоматического управления полётом БЛА. Эта задача предназначена для ознакомления с основными принципами автоматического регулирования.

Условие

В процессе полёта БЛА самолётного типа на него воздействуют различные аэродинамические эффекты. Так, например, компенсировав значение величины силы тяжести значением аэродинамической подъёмной силы, БЛА может совершать полёт на постоянной высоте. При этом разворот БЛА по углу курса достигается созданием угла крена, что приводит к появлению компоненты аэродинамической подъёмной силы, направленной горизонтально. В этом случае БЛА начинает курсовой разворот по дуге окружности, центр которой находится в том же направлении, куда направлена горизонтальная компонента подъёмной силы. Мгновенный радиус разворота при этом определяется соотношением:

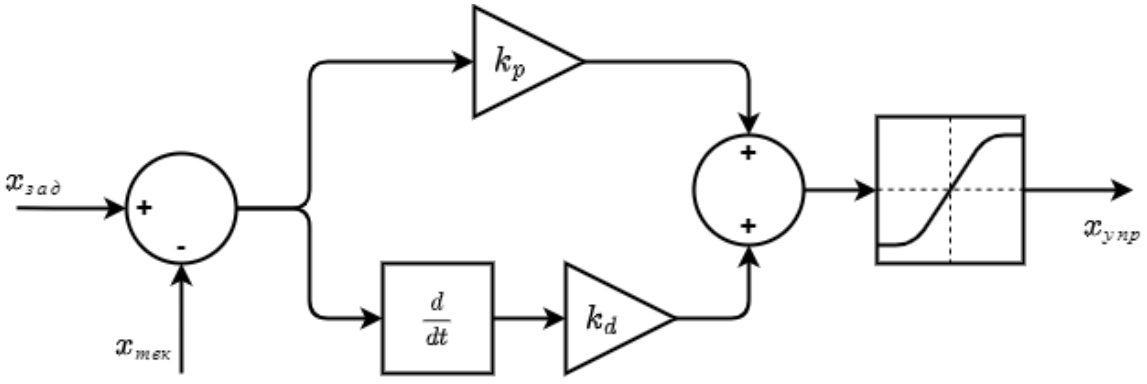
$$R = \frac{V^2}{g \cdot \tan \gamma},$$

где V — скорость полёта БЛА;

γ — угол крена;

g — ускорение свободного падения.

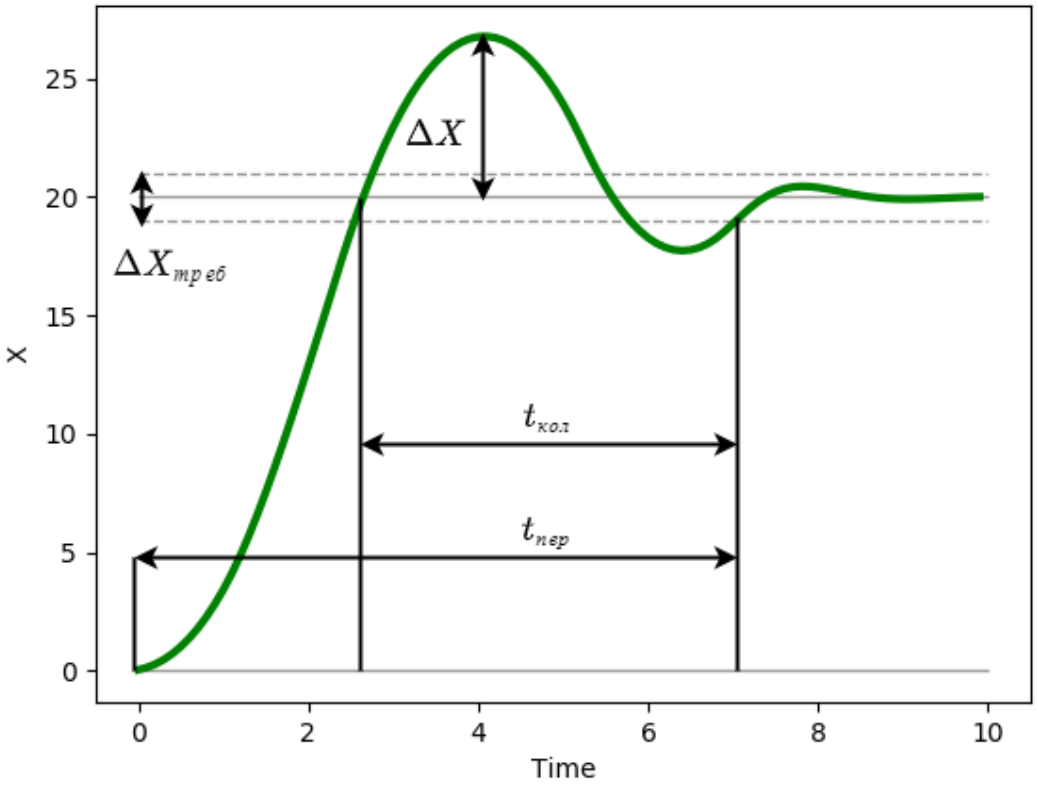
Система автоматического управления (САУ) БЛА в автоматическом режиме рассчитывает необходимый угол крена в каждый момент времени для осуществления разворота на заданный угол курса. Для плавного и эффективного регулирования положения БЛА в САУ применяются автоматические регуляторы. Одним из простейших регуляторов, часто встречающихся в бортовых САУ, является ПД (пропорционально-дифференциальный) регулятор.



Как показано на рисунке, на вход регулятора подаётся значение ошибки регулируемой величины (разница заданного и текущего значений). Эта ошибка поступает на вход двух звеньев: пропорционального и дифференциального. Пропорциональное звено умножает (усиливает) значение ошибки на некоторый постоянный коэффициент k_p, а дифференциальное звено умножает производную ошибки $\left(\frac{d}{dt}\right)$ на коэффициент k_d. Выходной сигнал регулятора, как правило, ограничивают в допустимых пределах (блок насыщения). Представленную структуру можно записать в виде формулы:

$$x_{\text{упр}} = (x_{\text{зад}} - x_{\text{тек}})k_p + \frac{d(x_{\text{зад}} - x_{\text{тек}})}{dt}k_d.$$

В общем виде переходный процесс регулируемой величины выглядит так.



На рисунке обозначены:

- Трубка точности $\Delta x_{\text{треб}}$ — допустимое граничное значение ошибки регулируемой величины.
- Время переходного процесса $t_{\text{пер}}$ — процесс считается завершённым, когда регулируемая величина попадает в заданную трубку точности и больше не выходит за её пределы.
- Время затухания $t_{\text{кол}}$ — время затухания колебаний после достижения требуемого значения (как правило, этот параметр зависит от величины дифференциального коэффициента k_d).
- Перерегулирование ΔX — отклонение от заданного значения величины в противоположную сторону (как правило, этот параметр зависит от величины пропорционального коэффициента k_p).

Напишите программу на языке Python, реализующую канал курса САУ БЛА, осуществляющего полёт на постоянной высоте. Входным значением этого канала управления должно быть заданные в каждый момент времени значения угла крена и скорости полёта, поступающие на вход функции моделирования полёта БЛА. Оценивается точность разворота БЛА по углу курса и время разворота (время переходного процесса).

Условия моделирования полёта БЛА:

- Шаг дискретизации $dt = 0,1$ с.
- Максимальный угол крена `roll_max` $= 20$ град.
- Максимальное угловая скорость крена `droll_max` $= 0,2$ рад/с.
- Минимальная линейная скорость полёта на постоянной высоте `velocity_min` $= 16$ м/с.
- Максимальная линейная скорость полёта на постоянной высоте `velocity_max` $= 32$ м/с.
- Максимальное линейное ускорение `dvelocity_max` $= 4$ м/с^2.
- Ускорение свободного падения $g = 9,81$ м/с^2.
- Начальное значение угла крена `roll` $= 0$ град.
- Начальное значение угла курса `heading` $= 0$ град.
- Начальное значение линейной скорости `velocity` $= 25$ м/с.

В шаблон программы включён скрытый класс UAV, реализующий моделирование согласно указанным параметрам. Код и структура класса (на языке python) имеют вид.

```
class UAV:

def __init__(self):
    self.heading = 0.0  # rad
    self.roll = 0.0  # rad
    self.g = 9.81  # m/s2
    self.velocity = 25.0  # m/s
    self.dt = 0.1  # sec
    self.droll = 0.0  # rad/sec
    self.dvelocity = 0.0  # m/s2
    self.elapsed = 0  # sec

def set_control(self, roll_value, vel_value):
    clamp = lambda n, minn, maxn: max(min(maxn, n), minn)
    # velocity channel
    vel_value = clamp(vel_value, 16, 32)
    err = vel_value - self.velocity
    dvel = 1.6 * err
    self.dvelocity = clamp(dvel, -4, 4)  # limit acceleration
    # roll channel
    roll_value = clamp(roll_value, -20 * math.pi / 180, 20 * math.pi / 180)
    err = roll_value - self.roll
    droll = 2.2 * err
    self.droll = clamp(droll, -0.2, 0.2)  # limit angular rate
    # Apply controls
    self.simulate()
```

```
def simulate(self):
    #
    # [HIDDEN]
    #
    self.heading += dheading * self.dt
    self.elapsed += self.dt

def get_velocity(self):
    return self.velocity

def get_roll(self):
    return self.roll

def get_heading(self):
    return self.heading

def get_elapsed(self):
    return self.elapsed
```

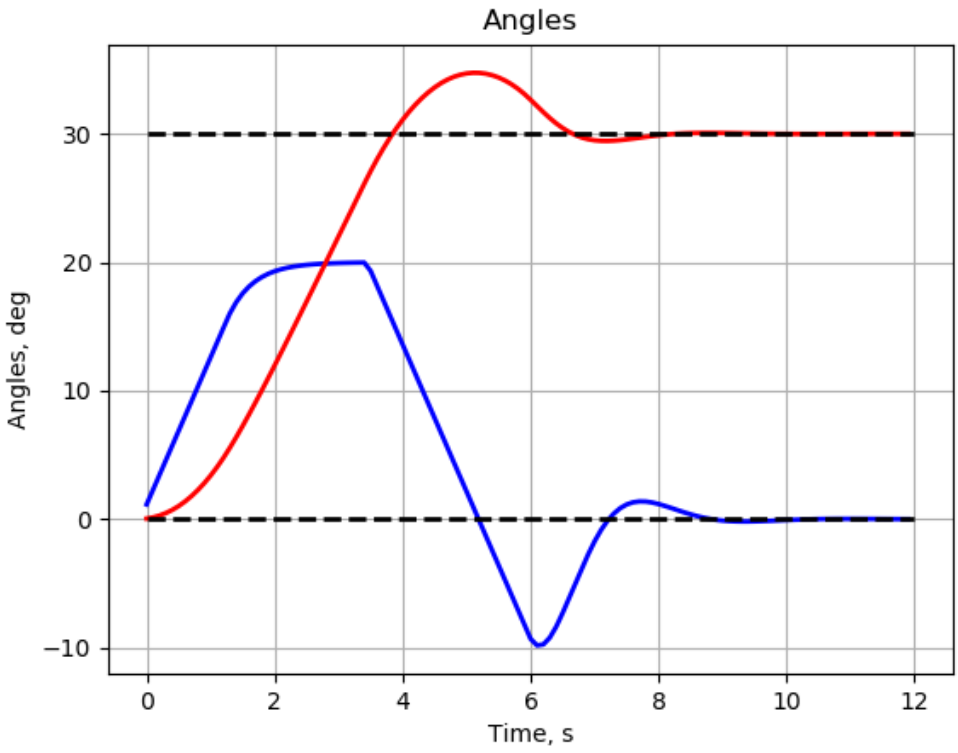
Все аргументы класса `UAV` являются закрытыми — вы не сможете получить доступ к ним. Аргумент elapsed содержит время моделирования. Его значение увеличивается на шаг моделирования dt при каждом запуске метода `simulate()`. Аргументы `droll`, `dvelocity` содержат текущие значения угловой скорости разворота по крену и линейного ускорения соответственно (эти значения определяются в методе `set_control()`).

Метод `simulate()` также является закрытым. Он осуществляет расчёт изменения угла крена, скорости полёта и угла курса БЛА на основе заданных параметров управления и реализацию соотношений для мгновенного радиуса разворота БЛА и угловой скорости разворота по курсу.

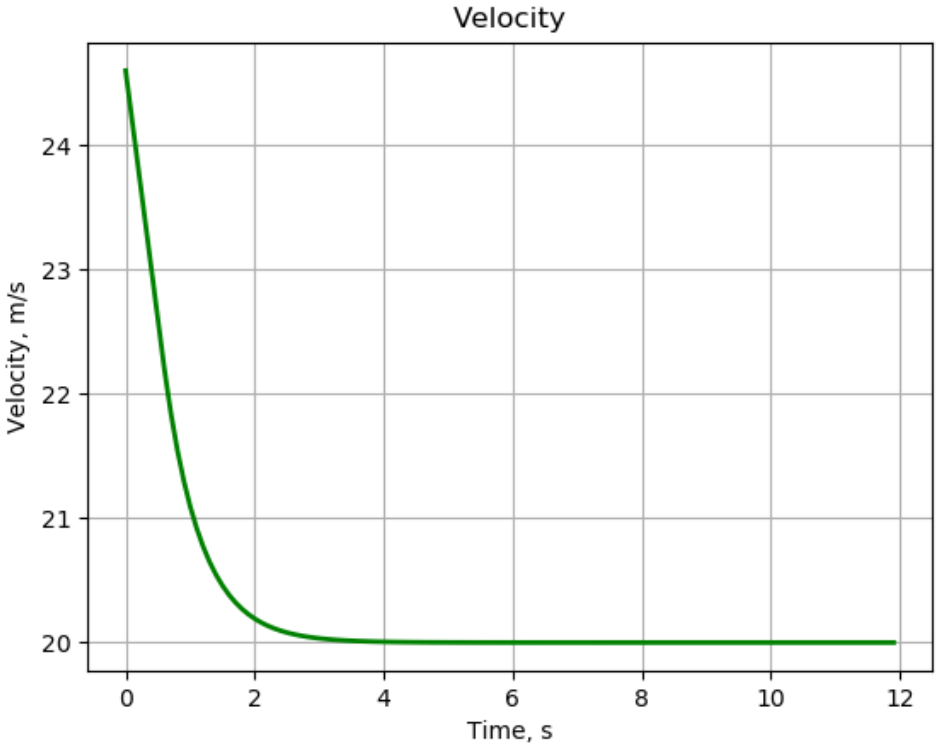
Открытые методы класса `UAV`:

- `get_elapsed()` — возвращает время полёта (в с);
- `get_heading()` — возвращает текущее значение угла курса (в рад);
- `get_roll()` — возвращает текущее значение угла крена (в рад);
- `get_velocity()` — возвращает текущее значение линейной скорости (в м/с);
- `set_control(roll_value, velocity_value)` — устанавливает заданные значения угла крена и скорости, рассчитывает `droll`, `dvelocity` и запускает метод `simulate()` для моделирования одного интервала dt полёта. Обратите внимание, что заданные значения устанавливаются не напрямую, а через $П$-регулятор (только пропорциональное звено). Это означает, что для достижения желаемого значения угла крена (например) БЛА понадобится какое-то время (время переходного процесса). Аргументы метода задаются в радианах и м/с соответственно.

Пример визуализации работы программы.



Углы крена (синяя линия) и курса (красная линия) при развороте на 30 град. по углу курса.



Линейная скорость при изменении скорости полёта на 5 м/с.

Эта задача имеет гибкий критерий оценки. Максимальное количество баллов,

которые можно получить за решение — 30. За каждые 0,2 с превышения времени эталонного переходного процесса и 0,1 град. ошибки установившегося значения угла курса будет сниматься один балл (таким образом, если превышение времени моделирования (по сравнению с эталонным переходным процессом) составило 0,1 с, а ошибка установившегося угла курса составила 0,09 град. — за решение задачи начисляется 30 баллов).

Обратите внимание, что если время вашего переходного процесса оказывается меньше эталонного, бонусные баллы за решение задачи не начисляются.

На платформе stepik вам доступно 4 открытых варианта задания и 2 закрытых. При отправке решения система автоматической проверки сообщит вам величины ошибок и ожидаемую оценку для каждого из вариантов. Итоговое оценивание будет осуществляться путём усреднения результата по двум закрытым вариантам.

Примечание: обратите внимание, что оценка выполнения не зависит от величины скорости полёта БЛА. Это означает, что в процессе моделирования вы можете выбирать любое значение скорости полёта из разрешённого диапазона [16, 32] м/с.

Шаблон программы уже включает в себя считывание данных из консоли, пожалуйста не меняйте имена системных переменных.

Формат входных данных

Входными данными являются требуемое значение угла курса (в град.) и время эталонного переходного процесса (в с) заданные через пробел:

```
HH T.T
```

Формат выходных данных

Шаблон программы автоматически выводит строку следующего вида:

```
Результаты моделирования: 0.024 0.098 6.2
```

В этой строке указаны установившееся значение угла курса (град.), угол крена на момент окончания моделирования (град.) и время моделирования (с). Решение засчитывается только в том случае, если переходный процесс был завершён (модуль угла крена меньше 0,1 град.).

Примеры

Пример №1

Стандартный ввод
`20 5.0`

Стандартный вывод
`Вход: 20.0 5.0` `Результаты моделирования: 20.071 0.069 5.0`

Ссылка на набор тестов: https://disk.yandex.ru/d/CvXpiu8vUKghNw.

Решение

1. Составить структуру П-Д регулятора.
2. Определить значения коэффициентов экспериментальным путём.
3. Рассчитать требуемый угол крена.
4. Остановить моделирование если выполняются требования окончания переходного процесса.

Пример программы-решения

Ниже представлено решение на языке Python 3.

```
uav = UAV()  # Не меняйте имя этой переменной, оно используется при выводе
↪  результата
dt = 0.1
print(heading_goal)  # Требуемое значение угла курса (град)
print(time_goal)  # Время эталонного переходного процесса (с)
# TODO: Впишите решение ниже этой строки
course = heading_goal * math.pi / 180.0
last_err = 0.0

for i in range(0, 200):
    err = course - uav.get_heading()
    if (abs(err) < 0.1 * math.pi / 180.0) and (abs(uav.get_norolld()) < 0.1):
        #uav.set_control(0, uav.get_velocity())
        break
    rc = 2.5 * err - 0.09 * last_err / dt
    last_err = err
    uav.set_control(rc, 17)
    #uav.set_control(10 * math.pi / 180, 16)
```