§2 Второй отборочный этап

Второй отборочный этап проводится в командном формате в сети интернет в рамках онлайн-симулятора конструирования полета космических аппаратов «Орбита». Продолжительность второго отборочного этапа — 2 месяца. Работы оцениваются автоматически средствами онлайн-симулятора. Задачи носят междисциплинарный характер и в более простой форме воссоздают инженерную задачу заключительного этапа, участники должны были писать программы полета на языке Python.

Решение каждой задачи дает определенное количество баллов. Некоторые задачи могут приносить разное количество баллов — в зависимости от качества и скорости их решения, ряд задач предполагает штрафы за число попыток.

Все условия задач доступны участникам с первого дня второго отборочного этапа. Команды могут выполнять задачи в любом порядке. Часть задач допускают неограниченное число попыток сдать решение (запусков), другие задачи предполагают штрафные баллы за превышение числа доступных попыток.

Задача 2.1. Баллистика

Условие:

Баллистическая ракета — ракета, получающая основной импульс от двигателей за короткое время разгона после старта и совершающая большую часть полета по неуправляемой баллистической траектории. Самый простой пример такого движения — камень, брошенный под углом к горизонту, за тем исключением, что начальный импульс ему сообщает рука, а не реактивный двигатель. Траекторией его полета в условиях пренебрежения сопротивлением воздуха будет парабола.

Нужная скорость и направление сообщаются баллистической ракете на активном участке полёта системой управления полётом ракеты. После отключения двигателя остаток пути полезная нагрузка ракеты движется по баллистической траектории. Баллистические ракеты могут быть многоступенчатыми, в этом случае после достижения заданной скорости отработавшие ступени отбрасываются. Такая схема позволяет уменьшить текущий вес ракеты, тем самым позволяя увеличить её скорость. Одноступенчатые ракеты, в свою очередь проще в конструировании и запуске. Вам предлагается запустить именно одноступенчатую ракету.

Постановка задачи

Вашей задачей в миссии «Баллистика» является запуск баллистической ракеты таким образом, чтобы она максимально точно попала в точку, удаленную на заданное расстояние (это расстояние будет дано в условии конкретной миссии) от точки запуска. Расстояние рассчитывается, как длина дуги, соединяющей точку старта и точку финиша. Для этого вы должны будете дополнить программу полета баллистической ракеты на языке Python значениями угла, под которым стартует ракета, и времени, через которое нужно выключить двигатель.

Ограничения модели

Упрощения и ограничения принятые в данной модели:

- Земля считается идеальным шаром радиусом $R_0 = 6371$ км.

- Мы рассматриваем полет в одной плоскости, проходящей через центр Земли. Т.е. у аппарата есть только две координаты.
- Мы пренебрегаем силой сопротивления воздуха.
- Аппарат представляет собой идеальный шар. Это довольно условное ограничение не приводит к каким-либо серьезным последствиям, так как сопротивлением воздуха в задаче мы пренебрегли.
- Мы пренебрегаем вращением Земли.
- Дискретность управления системами ракеты составляет 0,1 секунду. Это означает, что в программе полета аппарата все времена (они отсчитываются с момента старта) могут быть кратны не меньше, чем 0,1 с. При этом вы можете отдать две (и более) команды в одно и тоже время. Пример: вы не сможете выключить двигатель через 10,424 с. Это значение будет автоматически округлено до 10,4 с.

Параметры устройств и аппарата

Аппарат представляет из себя шар внешним радиусом R = 1,1 м. В этой миссии вы не сможете поменять радиус аппарата. Полная стартовая масса аппарата M = 3441,8 кг.
Используемые устройства:

Код	**Название**	**Описание**
CPU	Центральный компьютер	Центральный процессор, без него аппарат не будет работать
D	Диагностика базовая	Устройство, отвечающее за сбор и обработку телеметрии
Acm	Аккумулятор	Аккумулятор электроэнергии, без него не будут работать остальные устройства
ERD	Двигатель стартовый	Двигатель для взлета. Скорость истечения топлива: U = 4100 м/с, расход топлива: dm/dt = 70 кг/с (не регулируется)
FTL	Топливный бак	Баки для топлива. Емкость бака: 500 кг топлива.
T	Передатчик базовый	Радиопередатчик, без которого телеметрия не будет транслироваться

Приведены только необходимые для успешного выполнения задачи параметры устройств.

Критерии оценки

Оценка производится автоматизированными средствами тестирования.
Засчитываться будут попадания в область в диапазоне +-16% от заданного в условии значения.
Количество баллов, которые можно получить за решение задачи:

- **Первое попадание** — Первыми (по времени) попасть в цель (400 баллов)
- **Второе попадание** — Вторыми (по времени) попасть в цель (300 баллов)
- **Третье попадание** — Третьими (по времени) попасть в цель (200 баллов)
- **Точное наведение** — Попасть с точностью меньше 8% (300 баллов)
- **Оперативность развертывания** — Попасть в цель в первый день турнира (24 часа с открытия системы) (300 баллов)
- **Без ошибки** — Попасть с первой попытки (1000 баллов)
- **Пристрелка** — Попасть со второй или третьей попытки (600 баллов)
- **Есть попадание** — Попасть с четвертой попытки или больше (100 баллов)

Решение:

Задача про бросок тела под углом к горизонту хорошо известна всем, кто учился в школе. Известно и решение для дальности полета такого тела, если бросить его с поверхности Земли.

$$L = \frac{V^2 \sin 2\alpha}{g} \quad (1),$$

где V - начальная скорость тела, α - угол, под которым его бросили, а g - ускорение свободного падения вблизи поверхности Земли. Отсюда видно, что максимальная дальность полета при заданной скорости достигается, если запустить тело под углом 45° и равна:

$$L_{\max} = \frac{V^2}{g}.$$

Однако эта формула выведена и справедлива только при использовании нескольких приближений:

- Земля плоская;
- ускорение свободного падения постоянно по модулю и направлению, причем направленно строго перпендикулярно поверхности Земли;
- сопротивления воздуха нет.

Если с последним условием особых проблем нет, оно выполняется в условии нашей задачи, то первые два условиям задачи не соответствуют: Земля у нас круглая, а ускорение свободного падения направлено в центр Земли и, вообще говоря, зависит от высоты. Кроме того, мы не знаем начальную скорость аппарата, т.е. непосредственно использовать предложенную формулу нельзя, но можно получить ответ, укладывающийся в заданную погрешность, используя несколько приближений.

1) Сделаем Землю плоской:

Рассмотрим движение не вдоль дуги окружности, составляющей участок Земли, а вдоль прямой X, проведенной из точки старта перпендикулярно радиусу Земли к этой точке (см. рис 1). Если пренебречь изменением вектора g, то изменение координаты X вдоль такой оси будет описываться привычным законом.

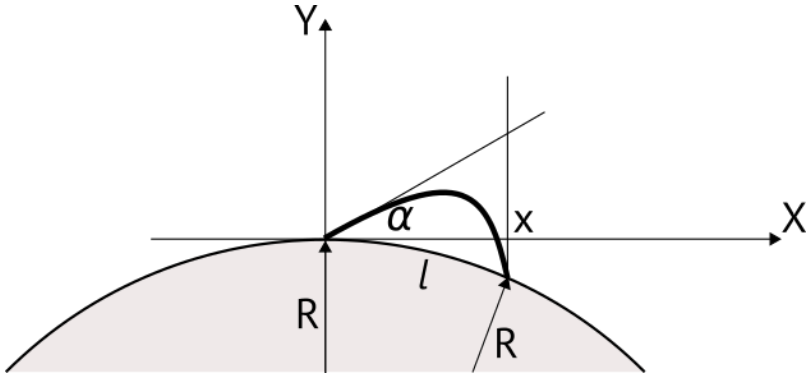


Рис. 1. Предлагаемая система координат

Из геометрии можно показать, что координата на этой оси и соответствующая длина дуги связаны следующим соотношением:

$$l = R \cdot \arcsin\frac{x}{R}$$

, где l - соответствующая длина дуги, x - координата вдоль предложенной оси, а R - радиус Земли.
Теперь можно использовать формулу

$$x = \frac{V^2 \sin 2\alpha}{g}$$

,
однако реальная дальность слегка увеличится из-за того, что проекция на предложенную ось всегда будет короче, чем длина дуги и слегка уменьшится из-за того, что g направлено не строго перпендикулярно оси X. Вы можете оценить вклад этих двух поправок в ответ.

2) Найдем начальную скорость:
Полет нашего аппарата состоит из двух основных этапов: полет с включенным двигателем и полет по баллистической траектории.

Силу тяги двигателя можно узнать по формуле P = Udm/dt.

Легко можно оценить максимальное время работы двигателя. Можно увидеть, что минимальное ускорение, сообщаемое двигателем составляет примерно 8g. То есть за время работы двигателя из-за силы тяжести направление движения ракеты изменится несущественно, и с хорошей точностью можно считать, что ракета все время летит по прямой под тем углом, под которым вы ее направили.

Тогда конечная скорость ракеты может быть оценена по формуле Циолковского:

$$V = U \ln\frac{M_н}{M_к}$$

, где $M_н$ — это полная масса аппарата до включения двигателя, а $M_к$ — после выключения двигателя.

В реальности скорость будет чуть ниже из-за того, что часть работы совершенной двигателями, пойдет на изменение не только кинетической, но и потенциальной энергии аппарата. Кроме того немного уменьшится и угол, под которым летит аппарат. Для более точного расчета попробуйте оценить эти поправки.

Конечная масса определяется расходом топлива, а он, в свою очередь, - временем отключения двигателя, которое Вы задаете соответствующей командой в программе полета.

3) Поправки баллистической траектории:
Дополнительно можно учесть, что в результате полета аппарата с включенными двигателями начальной точкой полета уже не будет поверхность Земли. Необходимо вычислить координаты точки начала баллистического полета и уже от нее считать дальность. При этом формулой (1) уже пользоваться нельзя, поскольку она выведена для случая полета тела с поверхности Земли.

Предложенная последовательность действий позволяет достаточно грубо (в пределах 10-20%) оценить дальность полета и нацелить свою ракету. Для более точной оценки необходимо вычислить упомянутые выше поправки или решить задачу точно.

Пример программы на языке Python, обеспечивающей решение задания для дальности 160 км:

```
t_off = 30
start_angle = 45

probe.engine_set_angle('ERD1', start_angle)
probe.set_device_state('ERD1', STATE_ON)
engine = True

probe.set_device_period('D1', 10)

while probe.run():
      t = probe.cpu_get_flight_time()
      if engine and t >= t_off:
      probe.set_device_state('ERD1', STATE_OFF)
      engine = False
      continue
```

Задача 2.2. Спутник связи "Молния"

Условие:

Миссия «Спутник связи "Молния"» моделирует ситуацию, в которой необходимо, имея ограниченный ресурс по топливу и параметрам спутника, организовать несколько сеансов продолжительной радиосвязи с наземным измерительным пунктом. Оказывается, что привычные круговые орбиты для решения задачи не подходят и нужно придумать, чем их заменить.

Постановка задачи

Ваш космический аппарат находится на круговой низкой орбите. Для выполнения миссии вам нужно провести два сеанса связи с наземным измерительным пунктом (НИП) длительностью не менее 8 часов с пропускной способностью канала не менее 1 Мб/с. Сеансом связи считается такое состояние спутника, когда у него непрерывно включен высокоскоростной передатчик, а НИП находится в зоне действия передатчика. Чтобы провести такой долгий сеанс связи, в ходе миссии вам придется перевести спутник на подходящую эллиптическую орбиту. Обратите внимание, что в данной миссии моделируется вращение Земли вместе с расположенными на ней НИПами. Полный оборот Земля совершает за 23 часа 56 минут.

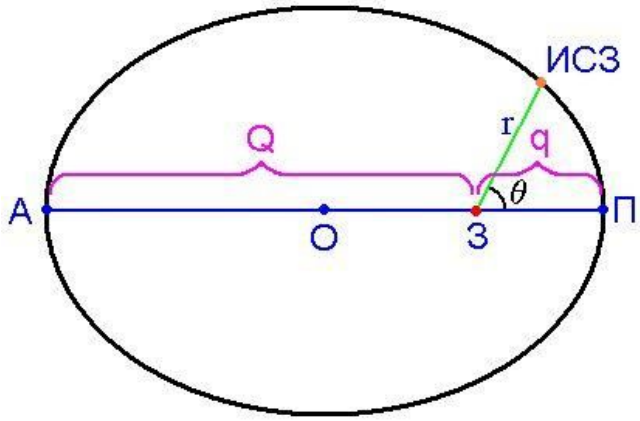
Самым простым решением для такой задачи был бы геостационарный спутник, находящийся над НИП, однако, в нашем случае, у спутника не хватит топлива для того, чтобы перейти с низкой орбиты, на которой он находится на старте выполнения задачи, сразу на геостационарную орбиту. Возможным решением этой проблемы является орбита «Молния», названная в честь советских спутников связи «Молния». Эта орбита представляет собой сильно вытянутую эллиптическую орбиту с апогеем порядка 40 тысяч километров и перигеем около 600 км, причем Земля находится в одном из фокусов этой орбиты. Поскольку полная энергия механическая энергия аппарата сохраняется, по мере удаления от Земли (т.е. приближения к апогею), скорость аппарата будет уменьшаться, и на максимальном удалении аппарат на такой орбите сможет достаточно долго находиться в зоне наблюдения НИП.

Эллиптическая орбита

Для решения задачи вам понадобится определить параметры орбиты, на которой должен находиться спутник, написать программу перехода на эту орбиту (в этом вам

помогут тренировочные миссии), суметь сориентировать спутник на НИП для успешной передачи.

Выбор орбиты определяется условием того, что ваш аппарат видит НИП непрерывно в течение 8 часов и, при этом, проходит по такой орбите как минимум дважды. Практически все круговые орбиты, которые доступны по запасу топлива для перехода вашему аппарату, слишком низки и аппарат потеряет НИП существенно быстрее, чем за 8 часов. Решением являются эллиптические орбиты, точные параметры одной из которых нужно подобрать.



Аппарат, движущийся по эллиптической орбите, вращается вокруг Земли, находящейся в одном из фокусов эллипса (точка З). Ближе всего к Земле он подлетит в перигее (точка П), а дальше всего в апогее (точка А). Эллипс определяется его большой полуосью (АО) и эксцентриситетом ($e = \sqrt{1 - \frac{b^2}{a^2}}$), где a - большая полуось эллипса, а b - малая полуось.

Используя законы Кеплера, можно получить следующее соотношение для расстояния до Земли и скорости аппарата в разных точках орбиты (в зависимости от угла θ):

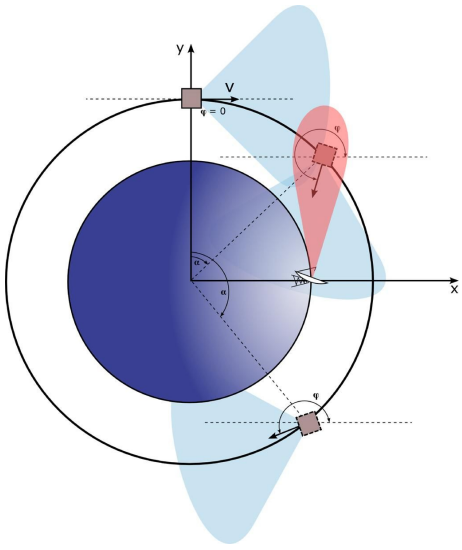
$$r(\theta) = \frac{a(2 - e^2)}{1 + e \cdot \cos\theta}, \quad V(\theta) = V_I \sqrt{\frac{2R}{r(\theta)} - \frac{R}{a}},$$

где $V_I = \sqrt{\frac{GM}{R}}$ = 7,91км/ч - первая космическая скорость для Земли. Период обращения спутника по эллиптической орбите может быть вычислен как

$$T = \left(\frac{2\pi R}{V_I}\right) \cdot \left(\frac{a}{R}\right)^{3/2}$$

Переход на эллиптическую орбиту

Новая эллиптическая орбита спутника будет определяться точкой, в которой находился спутник в момент, когда был включен двигатель, и его конечной скоростью.



Для перехода на эллиптическую орбиту можно использовать как двухимпульсный переход, подобный использованному в тренировочной миссии "Орбитальный маневр", так и одноимпульсный, рассчитать который значительно проще. Для неподвижной Земли такой переход надо было бы начинать в точке, диаметрально противоположной НИП, однако в нашей задаче Земля вращается, и

точку, в которой на самом деле нужно совершить переход, вам придется определить самостоятельно.

Приращение скорости можно рассчитать по формуле, приведенной в описании миссии «Орбитальный маневр» для первого перехода:

$$\Delta V = V_1\left(\sqrt{\frac{2R_2}{R_1+R_2}}-1\right)$$

, где R_1 - опорная орбита, на которой исходно находится ваш аппарат, а R_2 - расстояние от центра Земли до точки апогея вашей запланированной орбиты. Поскольку вам не нужно обратно выходить на круговую орбиту, второй переход не нужен. Дальнейшие рассуждения аналогичны приведенным в описании миссии «Орбитальный маневр».

Передатчик

Необходимо выбрать устройства, установленные на ваш спутник. В нашем случае главным таким устройством будет передатчик.

Необходимо передать на Землю определенный набор данных, обеспечив канал пропускной способностью 1Мб/с на 8 часов непрерывной передачи. Сеансом связи считается такое состояние спутника, когда у него непрерывно включен высокоскоростной передатчик, а НИП находится в зоне действия передатчика. Для этого необходим передатчик, обладающий определенными свойствами. Бортовая антенна передатчика закреплена на поверхности аппарата неподвижно и характеризуется диаграммой направленности, причем разные передатчики имеют разные диаграммы направленности и параметры пропускной способности канала передатчика (см. рисунок).

Пропускную способность передатчика можно рассчитать по формуле:

$$R = \frac{1}{100}\cdot\frac{G_1 G_2 P_1}{\left(\frac{4\pi\cdot L_{НИП}}{\lambda_1}\right)^2}\cdot\left(\frac{\log_2 M}{1.2kT_2}\right)$$

,

где R- скорость передачи информации в бит/с, P_1 - мощность передатчика аппарата, G_1 и G_2 - коэффициенты усиления бортовой и наземной антенн, которые зависят от типа антенны и заданы в параметрах подсистем (см. подсказку при выборе соответствующей подсистемы),

$$L_{нип} = \sqrt{(X_{НИП} - X_{апп})^2 + (Y_{НИП} - Y_{апп})^2 + (Z_{НИП} - Z_{апп})^2}$$

,

L - расстояние от аппарата до НИП в метрах, λ_1 - длина волны передатчика (ее можно определить через частоту передачи, заданную в параметре подсистемы связи), k $=1.38\cdot10^{-23}$ Вт·Гц$^{-1}$·К$^{-1}$ - постоянная Больцмана, T_2 = 1000K - шумовая температура приемника, M = 4 - кратность манипуляции с фазой. Коэффициент 1/100 возникает вследствие того, что отношение сигнал/шум не должно опускать ниже 100 в процессе связи.

Тепловой и энергетический баланс

После выбора типа передатчика необходимо определить алгоритм его включения, который в свою очередь зависит от ориентации аппарата на НИП. Самый простой способ – ориентировать свой аппарат на центр Земли. Для решения этого может быть

достаточно, однако если вам удастся сориентировать аппарат точно на НИП, вы получите дополнительные баллы.

После того, как основные параметры орбиты и аппарата рассчитаны и заданы, необходимо проверить энергетический и тепловой баланс аппарата. Необходимо учесть, что достаточно продолжительное время из-за вытянутой орбиты аппарат будет освещен Солнцем и может перегреться.

Критерии оценки

Оценка производится автоматизированными средствами тестирования, встроенными в симулятор полета космического аппарата.

- **Первые на орбите**: 2000 баллов - Выполнить миссию первыми
- **Вторые на орбите**: 1800 баллов - Выполнить миссию вторыми
- **Третьи на орбите**: 1600 баллов - Выполнить миссию третьими
- **Идеальный расчет**: 6000 баллов - Выполнить миссию с первой попытки
- **Высокая надежность**: 4000 баллов - Выполнить миссию со второй - пятой попытки
- **Миссия выполнена**: 2000 баллов - Выполнить миссию с шестой и более попы
- **Молниеносное развёртывание**: 5000 баллов - Провести три сеанса связи
- **Ни единого разрыва**: 10000 баллов - Длительность каждого сеанса не менее 10 часов

Решение

Конфигурация аппарата, обеспечивающей решение задания для начальной высоты 550 км и НИПа в положении 0 градусов:

- Запас топлива: 170 кг
- Напряжение в сети питания: 27 В
- Доля площади СБ на боковых гранях: 100%
- Доля площади радиатора на боковых гранях: 0%
- Доля площади радиатора на верхней и нижней гранях: 75%
- Корпус для миниспутника
- СОТР с очень большим нагревателем
- Навигатор-1
- Система ориентации с большим управляющим моментом
- Двигатель с большой тягой и очень большим баком
- БЦВМ-2
- СЭП с большим аккумулятором
- Система связи Ku-диапазона
- Телеметрия с ненаправленной антенной

Пример программы полета на языке Python:

```
initial_height = 550000.0
target_height = 40000000.0

ground_station_angle = 0.0

G = 6.6742e-11
earth_mass = 5.9726e24
earth_radius = 6371032.0
earth_rotation_period = 86164.1
```

```
max_orientation_torsion = 0.0165

max_engine_traction = 0.165
engine_specific_impulse = 3041.0

dry_mass = 40.0 + 25.0 + 6.6 + 0.6 + 0.4 + 3.0 + 32.0 + 1.5 + 7.5

heater_power = 70

angular_velocity_precision = 2e-3
orient_angle_precision = 5e-3
navig_angle_precision = 5e-3
coord_precision = 1e-2

def normalize_angle(angle):
     normalized_angle = angle
     while normalized_angle < 0:
     normalized_angle += 360
     while normalized_angle >= 360:
     normalized_angle -= 360
     return normalized_angle

def normalize_angle_difference(angle_difference):
     normalized_angle_difference = angle_difference
     while normalized_angle_difference < -180:
     normalized_angle_difference += 360
     while normalized_angle_difference >= 180:
     normalized_angle_difference -= 360
     return normalized_angle_difference

class DampingMode(object):
     def __init__(self):
     self.enabled = False
     self.active = False
     self.desired_angular_velocity = 0.0

     def run(self):
     if self.enabled:
          angular_velocity =
sputnik.orientation.get_angular_velocity(AXIS_Z)
          if abs(angular_velocity - self.desired_angular_velocity)
> angular_velocity_precision:
               if not self.active:
                    self.active = True
                    if angular_velocity >
self.desired_angular_velocity:
                    torsion = -max_orientation_torsion
                    else:
                    torsion = max_orientation_torsion
                    sputnik.orientation.start_motor(AXIS_Z)
                    sputnik.orientation.set_motor_moment(AXIS_Z,
torsion)
```

```
            elif self.active:
                  sputnik.orientation.stop_motor(AXIS_Z)
                  self.active = False

damping_mode = DampingMode()

class OrientationMode(object):
      def __init__(self):
      self.enabled = False
      self.active = False
      self.differential = True
      self.desired_angle = 0.0
      self.desired_angular_velocity = 0.0
      self.max_angular_acceleration = 0.0
      self.inertia_moment = 0.0

      def run(self):
      if self.enabled:
            angle = sputnik.orientation.get_angle(AXIS_Z)
            angular_velocity =
sputnik.orientation.get_angular_velocity(AXIS_Z)
            desired_angle_change =
normalize_angle_difference(self.desired_angle - angle)
            if ((abs(angular_velocity -
self.desired_angular_velocity) > angular_velocity_precision) or
                  (abs(desired_angle_change) >
orient_angle_precision)):
                  proportional_coefficient = 8 *
self.max_angular_acceleration / 180
                  differential_coefficient = 8 *
math.sqrt(self.max_angular_acceleration / 360)
                  angular_acceleration = desired_angle_change *
proportional_coefficient
                  if self.differential:
                        angular_acceleration -= (angular_velocity -
self.desired_angular_velocity) * differential_coefficient
                  torsion = self.inertia_moment * angular_acceleration
                  if torsion > max_orientation_torsion:
                        torsion = max_orientation_torsion
                  if torsion < -max_orientation_torsion:
                        torsion = -max_orientation_torsion
                  sputnik.orientation.set_motor_moment(AXIS_Z,
torsion)
                  if not self.active:
                        self.active = True
                        sputnik.orientation.start_motor(AXIS_Z)
            elif self.active:
                  sputnik.orientation.stop_motor(AXIS_Z)
                  self.active = False

orientation_mode = OrientationMode()

class PhasingMode(object):
```

```
     def __init__(self):
     self.enabled = False
     self.active = False
     self.angle_alignment = 0.0
     self.initial_nav_angular_velocity = math.degrees(math.sqrt((G *
earth_mass) / ((initial_height + earth_radius) ** 3)))

     def run(self):
     if self.enabled:
          nav_angle = sputnik.navigation.get_z_axis_angle()
          orientation_mode.desired_angle = normalize_angle(360 -
nav_angle)
          orientation_mode.desired_angular_velocity =
-self.initial_nav_angular_velocity
          if not self.active:
               self.active = True

phasing_mode = PhasingMode()

class TrackingMode(object):
     def __init__(self):
     self.enabled = False
     self.active = False
     self.prev_flight_time = None
     self.prev_nav_angle = None

     def run(self):
     if self.enabled:
          flight_time = sputnik.cpu.get_flight_time()
          nav_angle = sputnik.navigation.get_z_axis_angle()
          if self.prev_flight_time is not None:
               tick = flight_time - self.prev_flight_time
               desired_angular_velocity = -1.0 *
normalize_angle_difference(nav_angle - self.prev_nav_angle) / tick
          else:
               desired_angular_velocity = 0.0
          self.prev_flight_time = flight_time
          self.prev_nav_angle = nav_angle
          orientation_mode.desired_angle = normalize_angle(270 -
nav_angle)
          orientation_mode.desired_angular_velocity =
desired_angular_velocity
          if not self.active:
               self.active = True

tracking_mode = TrackingMode()


state = 0

heating = False

while sputnik.cpu.run():
```

```
      damping_mode.run()
      orientation_mode.run()
      phasing_mode.run()
      tracking_mode.run()

      temperature = sputnik.heat_control.get_temperature()
      if (temperature < 285) and not heating:
      heating = True
      sputnik.heat_control.set_power(heater_power)
      sputnik.heat_control.start_heating()
      elif (temperature > 300) and heating:
      heating = False
      sputnik.heat_control.set_power(0)
      sputnik.heat_control.stop_heating()

      if state == 0:
      sputnik.telemetry.set_state(STATE_ON)
      sputnik.navigation.set_state(STATE_ON)
      sputnik.orientation.set_state(STATE_ON)
      sputnik.heat_control.set_state(STATE_ON)
      initial_angular_velocity =
sputnik.orientation.get_angular_velocity(AXIS_Z)
      flight_time = sputnik.cpu.get_flight_time()
      stabilization_begin_time = flight_time
      damping_mode.enabled = True
      state = 1
      continue

      if (state == 1) and not damping_mode.active:
      damping_mode.enabled = False
      orientation_mode.max_angular_acceleration =
abs(initial_angular_velocity) / (sputnik.cpu.get_flight_time() -
stabilization_begin_time)
      orientation_mode.inertia_moment = max_orientation_torsion /
orientation_mode.max_angular_acceleration
      state = 10
      continue

      if state == 10:
      phasing_mode.enabled = True
      state = 11
      continue

      if (state == 11) and phasing_mode.active:
      orientation_mode.enabled = True
      state = 12
      continue

      if (state == 12) and not orientation_mode.active:
      sputnik.engine.set_state(STATE_ON)
      transfer_delta_v = abs(math.sqrt(G * earth_mass /
(initial_height + earth_radius)) * (math.sqrt(2 * (target_height +
```

```
earth_radius) / ((target_height + earth_radius) + (initial_height +
earth_radius))) - 1))
     transfer_start_fuel = sputnik.engine.get_fuel()
     sputnik.engine.set_state(STATE_OFF)
     transfer_end_fuel = ((dry_mass + transfer_start_fuel) /
math.exp(transfer_delta_v / engine_specific_impulse)) - dry_mass
     if transfer_end_fuel < 0.0:
          transfer_end_fuel = 0.0
     transfer_burn_time = (transfer_start_fuel - transfer_end_fuel)
/ max_engine_traction
     transfer_burn_angle = transfer_burn_time *
phasing_mode.initial_nav_angular_velocity
     transfer_delta_v = transfer_delta_v *
math.sqrt(math.radians(transfer_burn_angle) / (2 *
math.sin(math.radians(transfer_burn_angle / 2))))
     transfer_end_fuel = ((dry_mass + transfer_start_fuel) /
math.exp(transfer_delta_v / engine_specific_impulse)) - dry_mass
     if transfer_end_fuel < 0.0:
          transfer_end_fuel = 0.0
     transfer_burn_time = (transfer_start_fuel - transfer_end_fuel)
/ max_engine_traction
     transfer_burn_angle = transfer_burn_time *
phasing_mode.initial_nav_angular_velocity
     state = 13
     continue

     if state == 13:
     nav_angle = sputnik.navigation.get_z_axis_angle()
     start_angle = ground_station_angle + 180.0
     start_angle += 90.0
     start_angle += 360.0 * sputnik.cpu.get_flight_time() /
earth_rotation_period
     start_angle = normalize_angle(start_angle)
     if abs(normalize_angle_difference(nav_angle - start_angle)) <
transfer_burn_angle / 2:
          transfer_center_angle = nav_angle
          sputnik.engine.set_state(STATE_ON)
          sputnik.engine.set_traction(max_engine_traction)
          sputnik.engine.start_engine()
          state = 14
     continue

     if state == 14:
     fuel = sputnik.engine.get_fuel()
     if fuel <= transfer_end_fuel:
          sputnik.engine.stop_engine()
          sputnik.engine.set_state(STATE_OFF)
          phasing_mode.enabled = False
          #orientation_mode.desired_angle = normalize_angle(270.0 -
(ground_station_angle + 90.0))
          #orientation_mode.desired_angular_velocity = 0.0
          tracking_mode.enabled = True
          sputnik.transmitter.set_state(STATE_ON)
```

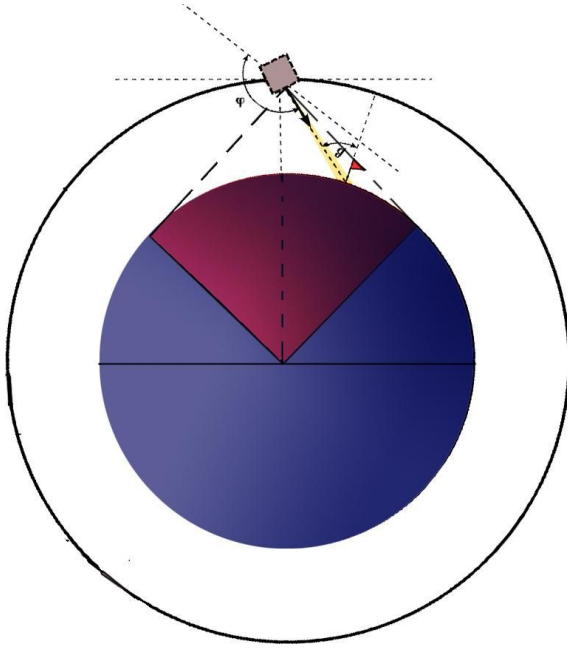
```
      state = 15
continue
```

Задача 2.3

Система предупреждения о ракетном нападении

Условие:

В современном мире, переполненном ядерным оружием, спутники осуществляют крайне важную функцию предупреждения запусков ракет. Во время разгона баллистическая ракета выделяет большое количество энергии и хорошо заметна в инфракрасном диапазоне. Находящийся на орбите спутник с помощью инфракрасной камеры может легко идентифицировать место старта ракеты, а это делает возможным перехват.



Постановка задачи

Ваш космический аппарат находится на геостационарной орбите. Его зона ответственности - сектор земной поверхности ±45 градусов от точки стояния. Из этого региона в ходе миссии будут запущены баллистические ракеты, которые необходимо обнаружить и перехватить. На активном участке полёта факел ракеты хорошо виден в ИК-диапазоне. Активный участок длится 180 секунд, за это время ракета достигает высоты около 160 км. Ваш аппарат должен производить круглосуточную съёмку Земли с помощью ИК-телескопа и оперативно передавать полученные данные на Землю. Для успешного перехвата ракеты необходимо передать на Землю снимок с её изображением не позднее, чем через 180 секунд после пуска. Для получения достаточно полного покрытия поверхности рекомендуется производить съёмку с угловой скоростью вращения аппарата не более 1°/с. Обратите внимание, что в данной миссии моделируется вращение Земли вместе с расположенными на ней НИПами. Полный оборот Земля совершает за 23 часа 56 минут.

Наблюдение

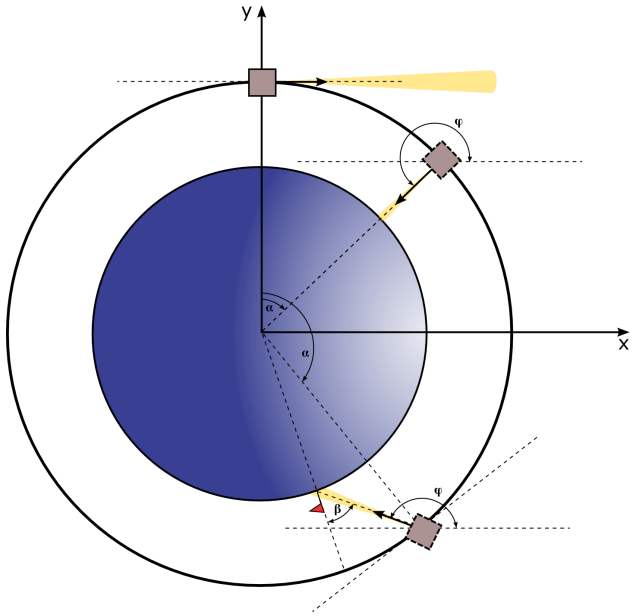
Обратите внимание на то, что аппарат не оборудован средствами обработки полученной информации, поэтому после каждого снимка необходимо передать его на Землю. С учетом времени, которое нужно на взлет ракеты, раз в 180 секунд ваш аппарат должен сканировать каждый участок подотчетной ему зоны и передать каждый полученный снимок в НИП. В данной задаче это означает, что необходимо сразу после съемки одного кадра переслать полученный снимок в НИП.

Ориентация аппарата

Ваш аппарат находится на геостационарной орбите. Это означает, что он всегда находится над одной точкой экватора, долгота которой называется точкой стояния. Высота такой орбиты составляет 35794 км. Чтобы камера спутника смотрела всегда на эту точку, после стабилизации аппарат необходимо ориентировать на нее, а затем поворачивать с угловой скоростью, совпадающей с угловой скоростью вращения аппарата вокруг Земли. Однако для успешного обнаружения ракет нужно смотреть не только на эту точку, но и на все точки поверхности Земли, долгота которых отличается от точки стояния на величину 45˚ как в большую, так и в меньшую сторону. На рисунке эта область отмечена красным.

Это означает, что на постоянный поворот накладывается дополнительный периодический поворот аппарата, параметры которого необходимо выбрать вам.

При этом вы не можете непосредственно изменить угловую скорость аппарата. Вы можете управлять только моментом импульса маховика. Напрашивающееся решение - задать момент импульса как функцию от времени - теоретически может быть и хорошим, но в реальных задачах, в частности, в этой, ошибки будут накапливаться слишком быстро. Правильным решением будет реализация системы с обратной связью. Например вы можете задать зависимость момента импульса маховика не от времени, а от угла и угловой скорости.



Инфракрасный телескоп

Для получения снимков Ваш аппарат должен быть оборудован ИК-телескопом. ИК-телескоп представляет собой один из объектов типа «камера».

Рассмотрим моделирование процесса получения изображений с помощью камеры. Камера устанавливается на аппарате таким образом, что ее ось совпадает с направлением ориентации аппарата – на поверхности 1 (см. рисунок).

Ориентация космического аппарата характеризуется углом φ. Камера расположена на поверхности 1, поэтому направлена под этим же углом. В нашей задаче важнее всего направить камеру в правильные регионы Земли.

Камера управляется через специальные команды в программе полета. Существует возможность сделать одиночный снимок или снимать поток кадров. Снимок, полученный камерой, должен быть целиком передан на Землю по высокопроизводительному каналу связи.

Еще один важный параметр камеры – объем оперативной памяти вычислительной системы. При включении камеры поток данных начинает непрерывно поступать в память вычислительной системы. Снимок отправляется в радиоканал после выключения камеры. Если оперативная память вычислительной системы камеры переполняется, существующее изображение сбрасывается. Так можно потерять удачный снимок.

Критерии оценки

Оценка проводится автоматизированными средствами тестирования, встроенными в симулятор полета космического аппарата.

- **Первые на орбите**: 2400 баллов - Выполнить миссию первыми
- **Вторые на орбите**: 2200 баллов - Выполнить миссию вторыми
- **Третьи на орбите**: 2000 баллов - Выполнить миссию третьими
- **Идеальный расчет**: 8000 баллов - Выполнить миссию с первой попытки
- **Высокая надежность**: 6000 баллов - Выполнить миссию со второй - пятой попытки
- **Миссия выполнена**: 3000 баллов - Выполнить миссию с шестой и более попытки
- **Ты не пройдёшь!**: 10000 очков - Перехватить все ракеты

Решение:

Конфигурация аппарата, обеспечивающей решение задания для начальной высоты 35794 км (ГСО):

- Топливо: 0 кг
- Напряжение: 27 В
- Доля площади СБ на боковых гранях: 100%
- Доля площади радиатора на боковых гранях: 0%
- Доля площади радиатора на верхней и нижней гранях: 75%
- Корпус для миниспутника
- СОТР с очень большим нагревателем
- Навигатор-1
- Система ориентации с очень большим управляющим моментом
- БЦВМ-2
- СЭП с очень большим аккумулятором
- Система связи Ku-диапазона
- Телеметрия с ненаправленной антенной
- ИК-телескоп

Пример программы полета на языке Python, решающей данную задачу:

```
initial_height = 35794000.0

ground_station_angle = 0.0

angle_offset = 6.0
scanning_period = 140.0

G = 6.6742e-11
earth_mass = 5.9726e24
earth_radius = 6371032.0
earth_rotation_period = 86164.1

max_orientation_torsion = 0.25

heater_power = 100

angular_velocity_precision = 2e-3
orient_angle_precision = 5e-3
navig_angle_precision = 5e-3
coord_precision = 1e-2

def normalize_angle(angle):
      normalized_angle = angle
      while normalized_angle < 0:
      normalized_angle += 360
      while normalized_angle >= 360:
      normalized_angle -= 360
      return normalized_angle

def normalize_angle_difference(angle_difference):
      normalized_angle_difference = angle_difference
      while normalized_angle_difference < -180:
      normalized_angle_difference += 360
```

```
      while normalized_angle_difference >= 180:
      normalized_angle_difference -= 360
      return normalized_angle_difference

class DampingMode(object):
      def __init__(self):
      self.enabled = False
      self.active = False
      self.desired_angular_velocity = 0.0

      def run(self):
      if self.enabled:
            angular_velocity =
sputnik.orientation.get_angular_velocity(AXIS_Z)
            if abs(angular_velocity - self.desired_angular_velocity)
> angular_velocity_precision:
                  if not self.active:
                        self.active = True
                        if angular_velocity >
self.desired_angular_velocity:
                        torsion = -max_orientation_torsion
                        else:
                        torsion = max_orientation_torsion
                        sputnik.orientation.start_motor(AXIS_Z)
                        sputnik.orientation.set_motor_moment(AXIS_Z,
torsion)
            elif self.active:
                  sputnik.orientation.stop_motor(AXIS_Z)
                  self.active = False

damping_mode = DampingMode()

class OrientationMode(object):
      def __init__(self):
      self.enabled = False
      self.active = False
      self.differential = True
      self.desired_angle = 0.0
      self.desired_angular_velocity = 0.0
      self.max_angular_acceleration = 0.0
      self.inertia_moment = 0.0

      def run(self):
      if self.enabled:
            angle = sputnik.orientation.get_angle(AXIS_Z)
            angular_velocity =
sputnik.orientation.get_angular_velocity(AXIS_Z)
            desired_angle_change =
normalize_angle_difference(self.desired_angle - angle)
            if ((abs(angular_velocity -
self.desired_angular_velocity) > angular_velocity_precision) or
                  (abs(desired_angle_change) >
orient_angle_precision)):
```

```
                proportional_coefficient = 8 *
self.max_angular_acceleration / 180
                differential_coefficient = 8 *
math.sqrt(self.max_angular_acceleration / 360)
                angular_acceleration = desired_angle_change *
proportional_coefficient
                if self.differential:
                     angular_acceleration -= (angular_velocity -
self.desired_angular_velocity) * differential_coefficient
                torsion = self.inertia_moment * angular_acceleration
                if torsion > max_orientation_torsion:
                     torsion = max_orientation_torsion
                if torsion < -max_orientation_torsion:
                     torsion = -max_orientation_torsion
                sputnik.orientation.set_motor_moment(AXIS_Z,
torsion)
                if not self.active:
                     self.active = True
                     sputnik.orientation.start_motor(AXIS_Z)
           elif self.active:
                sputnik.orientation.stop_motor(AXIS_Z)
                self.active = False

orientation_mode = OrientationMode()

class TrackingMode(object):
     def __init__(self):
     self.enabled = False
     self.active = False
     self.prev_flight_time = None
     self.prev_nav_angle = None

     def run(self):
     if self.enabled:
          flight_time = sputnik.cpu.get_flight_time()
          nav_angle = sputnik.navigation.get_z_axis_angle()
          if self.prev_flight_time is not None:
               tick = flight_time - self.prev_flight_time
               desired_angular_velocity = -1.0 *
normalize_angle_difference(nav_angle - self.prev_nav_angle) / tick
          else:
               desired_angular_velocity = 0.0
          self.prev_flight_time = flight_time
          self.prev_nav_angle = nav_angle
          #orientation_mode.desired_angle = normalize_angle(270 -
nav_angle)
          orientation_mode.desired_angle = normalize_angle((270 -
nav_angle) - angle_offset)
          orientation_mode.desired_angular_velocity =
desired_angular_velocity
          if not self.active:
               self.active = True
```

```
tracking_mode = TrackingMode()

class ScanningMode(object):
      def __init__(self):
      self.enabled = False
      self.active = False

      def run(self):
      if self.enabled:
            flight_time = sputnik.cpu.get_flight_time()
            angle = sputnik.orientation.get_angle(AXIS_Z)
            nav_angle = sputnik.navigation.get_z_axis_angle()
            angular_acceleration = (2 * math.pi / scanning_period) **
2 * normalize_angle_difference((270 - nav_angle) - angle)
            torsion = orientation_mode.inertia_moment *
angular_acceleration
            if torsion > max_orientation_torsion:
                  torsion = max_orientation_torsion
            if torsion < -max_orientation_torsion:
                  torsion = -max_orientation_torsion
            sputnik.orientation.set_motor_moment(AXIS_Z, torsion)
            if not self.active:
                  self.active = True
                  sputnik.orientation.start_motor(AXIS_Z)

scanning_mode = ScanningMode()


state = 0

heating = False

while sputnik.cpu.run():
      damping_mode.run()
      orientation_mode.run()
      tracking_mode.run()
      scanning_mode.run()

      temperature = sputnik.heat_control.get_temperature()
      if (temperature < 285) and not heating:
      heating = True
      sputnik.heat_control.set_power(heater_power)
      sputnik.heat_control.start_heating()
      elif (temperature > 300) and heating:
      heating = False
      sputnik.heat_control.set_power(0)
      sputnik.heat_control.stop_heating()

      if state == 0:
      sputnik.telemetry.set_state(STATE_ON)
      sputnik.navigation.set_state(STATE_ON)
      sputnik.orientation.set_state(STATE_ON)
      sputnik.heat_control.set_state(STATE_ON)
```

```
     initial_angular_velocity =
sputnik.orientation.get_angular_velocity(AXIS_Z)
     flight_time = sputnik.cpu.get_flight_time()
     stabilization_begin_time = flight_time
     damping_mode.enabled = True
     state = 1
     continue

     if (state == 1) and not damping_mode.active:
     damping_mode.enabled = False
     orientation_mode.max_angular_acceleration =
abs(initial_angular_velocity) / (sputnik.cpu.get_flight_time() -
stabilization_begin_time)
     orientation_mode.inertia_moment = max_orientation_torsion /
orientation_mode.max_angular_acceleration
     state = 10
     continue

     if state == 10:
     tracking_mode.enabled = True
     state = 11
     continue

     if (state == 11) and tracking_mode.active:
     orientation_mode.enabled = True
     state = 12
     continue

     if (state == 12) and not orientation_mode.active:
     tracking_mode.enabled = False
     orientation_mode.enabled = False
     scanning_mode.enabled = True
     state = 20
     continue

     if state == 20:
     slot = sputnik.camera.take_photo()
     if slot is not None:
          sputnik.transmitter.send_photo(slot)
     continue
```