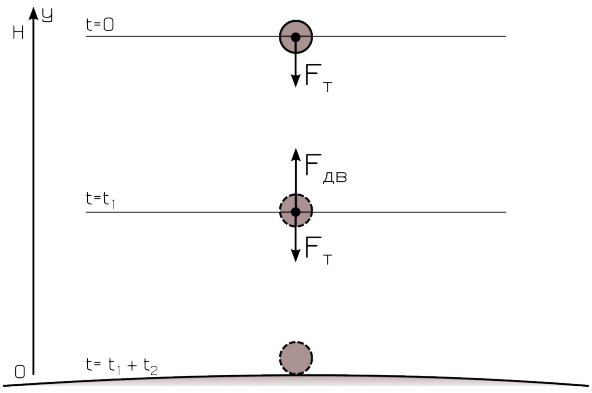
§2 Второй отборочный этап

Второй отборочный этап проводится в командном формате в сети интернет в рамках онлайн-симулятора полета и посадки космических аппаратов «Орбита». Продолжительность второго отборочного этапа — 2 недели. 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.

Объем и сложность задач этого этапа подобраны таким образом, чтобы решение всех задач одной командой было маловероятно. Это призвано обеспечить более осознанный выбор решаемых командой задач. 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. В данном этапе используется другая система оценки, теоретически можно получить суммарно от 0 до 74510 баллов.

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 «Посадка на Луну» (макс. 180 баллов)



Луна — ближайший к Земле астрономический объект. Посадка корабля на Луну — это самое простое задание, с которым человечество справлялось уже не раз.

Создание и запуск аппарата для исследования лунной поверхности состоит из нескольких этапов. В этой задаче мы мы рассмотрим только один, но самый интересный этап — посадку аппарата. Вам придется сконструировать собственный аппарат и составить техническое задание на его производство, дождаться результатов полета и получить телеметрию процесса посадки.

Успешно посадить аппарат с первой попытки не просто. В случае неудачи вам предстоит проанализировать данные телеметрии и изменить техническое задание для

следующего запуска.

2.1.1. Условия победы

За успешное решение этой миссии вы получаете баллы. В миссии возможны следующие достижения:

Первые на Луне: Первым сесть на Луну (40 балл.)

Вторые на Луне: Вторым сесть на Луну (30 балл.)

Третьи на Луне: Третьим сесть на Луну (20 балл.)

Идеальный расчет: Сесть с первой попытки (80 балл.)

Точный расчет: Сесть со второй попытки (70 балл.)

Есть посадка: Сесть с 3-10 попытки (50 балл.)

За ценой не постоим: Сесть с 11+ попытки (10 балл.)

Сроки соблюдены — Сесть на Луну за 24 часа с момента первого открытия задачи командой (30 балл.)

В яблочко! — Сесть на Луну с посадочной скоростью меньше 40 м/с (30 балл.)

2.1.2. Постановка задачи

Это только первая, тренировочная задача, поэтому в ней есть несколько допущений: аппарат падает вертикально на поверхность Луны, его начальная скорость равна нулю, а из доступного оборудования есть только демпфер и тормозной двигатель.

Задача состоит в том, чтобы определить, в какой момент времени t_1 нужно включить тормозной двигатель, чтобы к моменту посадки t_1+t_2 скорость корабля была бы меньше 50 м/с, иначе удар не удастся самортизировать с помощью демпфера.

Другими словами, вам нужно вычислить два параметра — время включения тормозного двигателя и время его выключения — и вставить их в программу полета аппарата.

Все исходные данные известны: это начальная высота, масса и радиус Луны, масса аппарата, сила тормозного двигателя. Исходные данные представлены в таблице далее.

2.1.3. Исходные данные

Параметр	Пояснение	Значение
H	Высота схода аппарата с орбиты (начало падения)	См. вариант на сайте
G	Гравитационная постоянная	$6{,}6742 \cdot 10^{-11}$ Н м^2/кг^2
M	Масса Луны	$7{,}35 \cdot 10^{22}$ кг
R	Радиус Луны	1 737 100 м (1737 км)
a_e	Ускорение, сообщаемое аппарату тормозным двигателем	Вычисляется по формуле: $a_e = F_{дв}/m$ (второй закон Ньютона), м/с^2
m	Масса аппарата	До включения двигателя масса равна 2935,88 кг
F_e или $F_{дв}$	Сила тяги тормозного двигателя, используемого на	Вычисляется по формуле: $F = u \times \Delta m$

	данном аппарате	
u	Скорость истечения газов из сопла двигателя	3600 м/с
Δm	Расход топлива	4,2 кг/с Всего аппарат содержит m_f = 500 кг топлива.

2.1.4. Аналитическое решение

Задача может быть решена аналитически. Для этого необходимы два допущения:

1. что масса корабля остается постоянной (топливо не расходуется);
2. что сила тяжести также не изменяется с высотой (стартовая высота мала по сравнению с радиусом Луны).

Хотя решение будет не очень точным, в этом случае легко записать систему уравнений и решить ее. Время включения двигателя (t_1) и время работы двигателя (t_2) можно вычислить следующим образом:

$$t_1 = \sqrt{2H(\frac{1}{g} - \frac{1}{a_e})} = \sqrt{2H(\frac{r^2}{GM} - \frac{m}{F_e})}, \quad t_2 = \frac{gt_1}{a},$$

где $a = a_e - g$, где *r* - высота аппарата, которая меняется в интервале от *(H + R)* до *R*, а *m* - масса аппарата, которая в ходе полета может уменьшиться до величины *(m — m_f)* в результате расхода топлива. Выберите значения *r* и *m* для ваших массы аппарата и стартовой высоты. Получите по формуле значения t_1 и t_2.

2.1.5. Программа полета

Вам нужно будет исправить программу полета и ввести туда рассчитанные значения времени включения двигателей (t_1) и времени работы двигателя (t_2):

```
t1 = # ПОСЧИТАТЬ И ВВЕСТИ t1
t2 = # ПОСЧИТАТЬ И ВВЕСТИ t2
engine = False
probe.set_device_period('D1', 10)
while probe.run():
    if not engine and t1 <= probe.cpu_get_flight_time() < t1 + t2:
        probe.set_device_state('EG1', STATE_ON)
        engine = True
        continue
    if engine and probe.cpu_get_flight_time() >= t1 + t2:
        probe.set_device_state('EG1', STATE_OFF)
        engine = False
```

```
            continue
        if probe.navigation_has_landed():
            # УСПЕШНАЯ ПОСАДКА
            break
```

2.1.6. Анализ телеметрии

По итогам запуска вам будут доступны записи телеметрии аппарата. Помимо общих сведений об аппарате, вы увидите таблицу изменений основных параметров аппарата с течением времени:

```
Ti=00:04:50 H=000041.4 Vy=-0041.6 Ac=04.15 Ae=005.8
```

Возможные параметры телеметрии представлены в таблице:

Название	Код	Единица измерения	Комментарий
Время полета	Ti	час:мин:сек	Период телеметрии 10 сек
Высота над поверхностью	H	м	Начальная высота 50 000 м
Вертикальная скорость	Vy	м/с	Начальная скорость равна 0. Напоминаем, что ось y направлена вверх
Суммарное ускорение	Ac	м/с^2	Абсолютное значение ускорения
Ускорение от двигателей	Ae	м/с^2	Абсолютное значение ускорения, создаваемого включенным двигателем

2.1.7. Переход от аналитического решения к фактическому

Предложенное аналитическое решение не дает точного правильного ответа: для некоторых значений t_1 аппарат разбивается. Это связано с тем, что в реальности изменениями массы и высоты аппарата нельзя пренебрегать. Вернемся к формуле:

$$t_1 = \sqrt{2H(\frac{r^2}{GM} - \frac{m}{F_e})}$$

Из нее видно, что в течение полета уменьшаются оба числителя. Можно оценить крайние значения t_1, а также значения, получаемые для средних величин r и m. Правильный ответ будет лежать в этой области.

Используйте результат первого полета, чтобы понять, должны ли вы уменьшить или увеличить время включения тормозного двигателя. Ориентируйтесь на диапазон значений t_1, полученных из аналитического решения. Посадите корабль на Луну за минимальное число попыток.

РЕШЕНИЕ:

Пример программы полета, аккуратно реализующей данный алгоритм для стартовой

высоты 50000 м:

```
01. engine = False
02. while probe.run():
03.     t = probe.cpu_get_flight_time()
04.     if not engine and 214 <= t <= 290:
05.         probe.set_device_state('EG1', STATE_ON)
06.         engine = True
07.         continue
08.     if engine and t >= 291:
09.         probe.set_device_state('EG1', STATE_OFF)
10.         engine = False
11.         continue
12.     if probe.navigation_has_landed():
13.         probe.telemetry_send_message('landed\n')
14.         break
```

Задача 2.2 «Посадка на Марс» (макс. 1650 баллов)

Красная Планета — намного более сложный объект для посадки космического аппарата, чем Луна. Во-первых, Марс намного массивнее, а значит сила тяжести играет куда большую роль. Во-вторых, на Марсе есть атмосфера, так что влияние сопротивления атмосферы на движение корабля около поверхности будет значительным.

В этой задаче также не будет рассматриваться работа аппарата на поверхности. В вашем распоряжении снова будет полностью сконструированный аппарат, но вам придется самостоятельно запрограммировать его полет: выбрать, в какой момент нужно будет включить тормозной двигатель, открыть парашют и т. д.

Анализ телеметрии позволит исправить ошибки, допущенные при посадке, уже в следующем аппарате (возможные параметры телеметрии указаны далее).

2.2.1. Условия победы

За успешное решение этой миссии вы получаете победные баллы. В миссии возможны следующие достижения:

Первые на Марсе — Первыми сесть на Марс (300 балл.)

Вторые на Марсе — Вторыми сесть на Марс (280 балл.)

Третьи на Марсе — Третьими сесть на Марс (250 балл.)

Точный расчет — Сесть с 1-3 попытки (600 балл.)

Есть посадка — Сесть с 3-15 попытки (450 балл.)

Воля к победе — Сесть с 16+ попытки (200 балл.)

Самые эффективные — Сесть на Марс за наименьшее число попыток (300 балл.)

Вторые по эффективности — Второе место по количеству попыток (280 балл.)

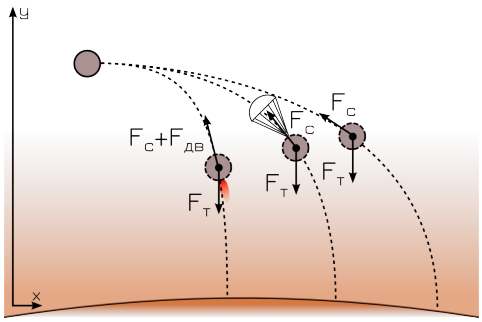
Третьи по эффективности — Третье место по количеству попыток (240 балл.)

Стартовое окно — Сесть на Марс в течение первых трех суток с момента первого открытия задачи командой (200 балл.)

Мягкая посадка — Сесть на Марс с посадочной скоростью меньше **15 м/с** (250 балл.)

2.2.2. Постановка задачи

По сравнению с Луной задача усложняется: теперь придется работать в двух измерениях. У аппарата есть начальная горизонтальная (орбитальная) скорость. К тому же, теперь на аппарат действует не только сила тяжести, но и сила аэродинамического сопротивления (Стокса), пропорциональная квадрату скорости аппарата.



Однако, и в этой задаче есть упрощение: поверхность планеты принимается за плоскость. Также вам будет доступна специальная программа для расчетов.

Условие задачи делает аналитическое решение очень сложным, поэтому мы предлагаем вам качественно оценивать значения скоростей и сил, а также тщательно анализировать результаты неудачных полетов. Все необходимые для расчетов параметры представлены в таблице далее.

2.2.3. Исходные данные

Параметр	Пояснение	Значение
G	Гравитационная постоянная	$6{,}6742 \cdot 10^{-11}$ Н м^2/кг^2
M	Масса Марса	$6{,}4185 \cdot 10^{23}$ кг
R	Радиус Марса	3 389 500 м
H	Высота стартовой орбиты	См. условие на сайте
F_c	Сила аэродинамического сопротивления	вычисляется по формуле $F_c = \frac{k\rho(y)v^2S}{2}$
k	Аэродинамический коэффициент. Известен для сферического аппарата	0,47
ρ(y)	Плотность атмосферы в зависимости от высоты	см. график изменения плотности в Приложении
S	Характерная площадь тела: равна площади сечения и увеличивается на площадь парашюта, если он раскрыт.	вычисляется по формуле $S = \pi r^2$, где **r** - радиус аппарата
r	Радиус аппарата	0,811 м
m	Масса аппарата (на начало полета)	1277,61 кг
$\mathbf{v_x}$	Стартовая горизонтальная скорость	3555,07 м/с
$\mathbf{v_y}$	Стартовая вертикальная скорость	0 м/с
$F_{дв}$ или F_e	Сила тяги посадочного двигателя	вычисляется по формуле $F = u \times \Delta m$

u	Скорость истечения газов из сопла двигателя	3600 м/с
Δm	Расход топлива	2 кг/с, аппарат снабжен 50 кг топлива
v_{Dm}	Предельная скорость для амортизации демпфером	40 м/с

Аппарат оснащен несколькими устройствами. В таблице представлены устройства, необходимы для успешной посадки:

Название	Код	Назначение
Аэродинамический тормоз	Hsl1	Применяется при входе в атмосферу. Этот массивный щит используется для гашения основной скорости аппарата.
Парашют для разреженной атмосферы	Pam1	Огромный по площади парашют позволяет замедлить аппарат даже в разреженной атмосфере.
Посадочный двигатель	EL1	Двигатель с небольшой тягой, позволяющий скорректировать скорость посадки аппарата.
Демпфер	Dm1	Демпфер позволяет амортизировать лишнюю скорость при достижении поверхности.

Составляя программу полета, каждое устройство можно **включить** или **выключить** (с помощью метода `set_device_sate` с параметрами `STATE_ON` и `STATE_OFF`), указав соответственно время включения или выключения в секундах от начала этапа**.**

Учтите, что:

1. Парашют имеет такой параметр, как **предельная скорость**: если парашют открыть слишком рано, его просто сорвет.
2. При отключении парашюта или аэродинамического тормоза они **отбрасываются**, а значит, их масса вычитается из массы аппарата. По этой причине не получится включить и выключить эти устройства несколько раз.
3. **Нельзя** открывать парашют до того, как был отброшен аэродинамический тормоз - они запутаются, и их сорвет.
4. Парашют или аэродинамический тормоз можно использовать вместе с двигателем.
5. Посадочный двигатель расходует топливо. Аппарат снабжен топливным баком на **50 кг топлива**.
6. При посадке аппарата, двигатель отключается автоматически.

Еще одно ограничение, которое появляется на Марсе в связи с посадкой в атмосфере - предельная перегрузка аппарата. Если ускорение аппарата превысит **155 м/с^2**, он будет разрушен перегрузкой.

Необходимые параметры парашюта и аэродинамического тормоза:

Название	Код	Площадь, м^2	Масса, кг	Предельная скорость, м/с
Аэродинамический тормоз	Hsl1	18	150	7000
Парашют для разреженной атмосферы	Pam1	200	10	600

Ограничения физической модели, которая используется при посадке на Марс указаны далее.

2.2.4. Разбираемся с программой для баллистических расчетов

Для посадки на Марс нам придется много использовать программу для расчетов на Земле. Программа для баллистических расчетов позволяет получить траекторию свободного падения тела с заданными параметрами на поверхность Марса (с учетом используемой физической модели гравитации, атмосферы и т.п.).

Конструкция аппарата фиксирована, поэтому начнем с известных параметров, например:

Характерная площадь тела: 2,0663 м^2

Масса: 1397,61 кг

Стартовое положение (x): 0 м

Стартовая высота (y): 80 000 м

Стартовая горизонтальная скорость (Vx): 3555,0733 м/с

Стартовая вертикальная скорость (Vy): 0 м/с

Введите известные начальные значения в программу для расчетов и определите скорость аппарата в момент касания поверхности. Будет ли данная скорость скомпенсирована демпфером?

Свободное падения корабля приводит к аварии. Что же делать?

Посмотрим на доступное оборудование - на самом деле, с момента старта у включен тепловой щит (Hsl1). Наш расчет этого не учитывал, ведь тепловой щит увеличивает эффективную площадь на 18 м^2.

Посчитайте новую характерную площадь тела как сумму площади аппарата и площади теплового щита. Как изменилась вычисляемая скорость посадки? Достаточно ли только теплового щита и демпфера для успешной посадки?

2.2.5. Что использовать - двигатель или парашют?

Чтобы еще как-то замедлить спуск, нужно посмотреть на оборудование корабля. Из подходящего оборудования у нас есть двигатели и парашют.

Двигатели характеризуются скоростью реактивной струи (в данном случае **3600 м/с**) и мощностью, которая пропорциональна расходу топлива (в нашем случае **2 кг/с**). Проведем

грубую оценку нашей возможности замедлить падение двигателями.

Используем формулу Циолковского для скорости ракеты:

$$v = u\,ln\left(\frac{M_1}{M_2}\right)$$

где u - удельная тяга (скорость истечения струи), M_1 - масса аппарата с топливом, M_2 - масса аппарата без топлива.

Подставим наши значения (M_2 получим путем вычитания массы топлива из общей массы) и получим скорость, которую мы будем вырабатывать двигателем.

Сможет ли вычисленная по формуле скорость, получаемая за счет двигателей, скомпенсировать разгон аппарата силой тяжести? (В реальности пользы от двигателя будет еще меньше, так есть сопротивление среды и другие факторы).

Если двигатели позволяют скомпенсировать скорость падения (которую мы вычислили на предыдущем шаге), надо использовать их при посадке. Иначе — использовать парашют для дальнейшего снижения скорости.

2.2.6. Посадка с парашютом

Как видно из конструкции корабля, на нем установлен Парашют для разреженной атмосферы (Pam1). Нам важны следующие его характеристики: площадь (**200 м^2**) и максимальная допустимая скорость (**600 м/с**). Как работать с площадью - понятно. Это величина, которая прибавляется к характерной площади тела и влияет на силу аэродинамического сопротивления нашего аппарата.

Предельная скорость - это та скорость которую выдерживает парашют. Если скорость потока воздуха (она же скорость нашего аппарата) будет больше, то парашют сорвет.

Итак, когда же раскрывать парашют? С одной стороны, чем раньше - тем лучше, тем больше времени он будет нас тормозить и тем сильнее снизится скорость. С другой стороны, если открыть его на слишком большой скорости, его сорвет потоком.

Посмотрим на нашу программу расчетов. Скорость аппарата складывается из скоростей V_x и V_y. Результирующую скорость V найдем по теореме Пифагора.

Используйте программу для расчетов, чтобы вычислить, на какой минуте будет достигнута приемлемая для выпуска парашюта скорость?

Так мы получим момент времени, в который можно сбрасывать тепловой щит и, **одновременно**, выпускать парашют. **Внимание:** если вы раскроете парашют хотя бы через секунду после отключения аэродинамического тормоза, аппарат может разогнаться до такой скорости.

Снова используйте программу для расчетов. Но в этот раз рассчитайте последний участок траектории - спуск на парашюте. В качестве начальных условий (скорость, высота) нужно взять результат из предыдущего запуска для того момента времени, когда мы выпускаем парашют. С какой теперь скорость будет происходить касание поверхности планеты?

Не забывайте, что парашют и тепловой щит не могут быть активны одновременно, а также то, что масса аппарата уменьшится на массу отброшенного щита.

Если скорости в момент касания не достаточно для компенсации демпфером, можно ли ее компенсировать двигателями? Для этой оценки у нас есть предельная скорость, полученная на предыдущем этапе.

2.2.7. Финальное торможение

Теперь нам предстоит решить вопрос, когда включать (и выключать) двигатели.

Если мы включим их слишком рано, корабль остановится на большой высоте, а потом когда кончится топливо упадет и разобьется. Если включим поздно, то двигатели не успеют затормозить корабль.

Точно математически решить эту задачу чрезвычайно сложно, так как движение нашего аппарата с учетом всех действующих на него сил описывается сложным дифференциальным уравнением. Но можно провести некоторые оценки. Для этого снова воспользуемся формулой Циолковского для реактивного движения.

Поскольку мы знаем, какую скорость нам надо погасить (это та скорость на которой мы столкнемся с поверхностью, если не включим двигатель), то мы можем посчитать массу топлива которую нам нужно для этого потратить.

$$v = u\,ln\left(\frac{M_1}{M_2}\right) \qquad \frac{v}{u} = ln\left(\frac{M_1}{M_2}\right) \qquad e^{\frac{v}{u}} = \frac{M_1}{M_2}$$

Зная расход топлива в секунду, мы можем посчитать длительность тормозного импульса. Например, если длительность получится 15 секунд, то включать двигатель нужно примерно за 10-14 секунд до предполагаемого приземления (поскольку по мере падения скорости время снижения будет расти).

Используя формулу Циолковского, оцените, в какой момент нужно включить тормозной двигатель. Теперь можно конструировать корабль и отправляться к Марсу.

2.2.8. Физическая модель

Аппарат приближается к Марсу на известной высоте (например, 80 км над

поверхностью планеты) с известной скоростью - первой космической, которая в свою очередь определяется по следующей формуле:

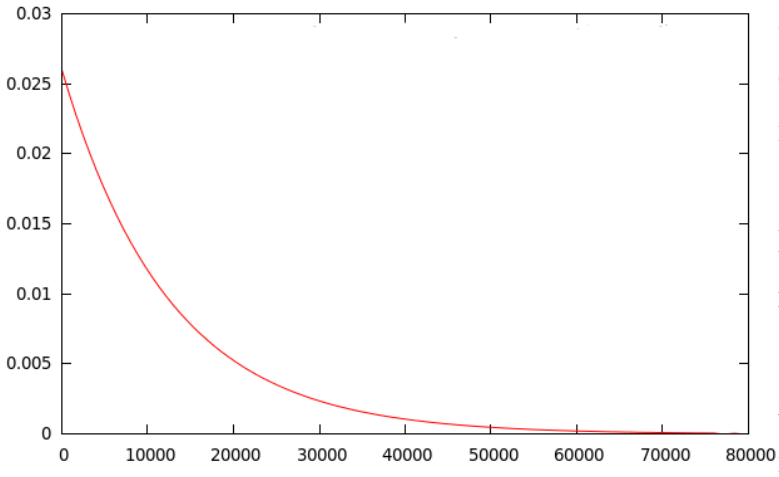
$v = \sqrt{\frac{G \cdot M}{R}}$, где G - гравитационная постоянная, M и R - масса и радиус Марса.

В целях упрощения мы рассматриваем только двумерную задачу с переходом от окружности к плоской поверхности, так что скорость раскладывается на две перпендикулярные компоненты. В начале вертикальная компонента скорости (v_y) равна нулю. Изменение величины ускорения аппарата определяется следующими уравнениями:

$$\bar{a} = \bar{g} - \frac{k\rho(y)Sv\bar{v}}{2m} - \frac{F_e\bar{v}}{mv}, \quad g = \frac{GM}{(R+y)^2}$$

где v — скорость аппарата, y — высота над поверхностью Марса, $\rho(y)$ — плотность атмосферы Марса в зависимости от высоты (экспоненциальная зависимость), k — аэродинамический коэффициент (аппарат имеет сферическую форму, он известен), S — площадь сечения аппарата, m — масса аппарата, F_e — сила тормозных двигателей.

Видно, что в общем случае на аппарат воздействуют только три силы (гравитационная, аэродинамического сопротивления по Стоксу и двигателей), причем аэродинамическое сопротивление и сила тяги двигателей направлены всегда против вектора скорости.



Плотность атмосферы Марса представлена на следующем графике.

Плотность меняется по экспоненте от 0,026 кг/м^3 на поверхности до 4,32 * 10^{-6} кг/м^3 на высоте 80 000 м.

2.2.9. Анализ телеметрии

После запуска аппарата, вам становится доступна запись его телеметрии. Есть два типа телеметрии: базовая и расширенная. Базовая телеметрия в процессе полета содержит таблицу изменений основных параметров с течением времени. Например:

```
Ti=00:03:17 X=0616595 H=016504.3 Vx=2371.1 Vy=-0590.0 Ac=13.74 Ae=000.0 As=016.2
```

Возможные параметры базовой телеметрии представлены в таблице:

Название	Код	Единица измерения	Комментарий

Время полета	Ti	час:мин:сек	Период телеметрии 1 сек
Горизонтальное положение	X	м	Начальное положение — 0 м
Высота над поверхностью	H	м	Начальная высота указана в условии
Горизонтальная скорость	Vx	м/с	Начальная скорость равна первой космической для Марса
Вертикальная скорость	Vy	м/с	Начальная скорость равна 0. Напоминаем, что ось **y** направлена вверх
Суммарное ускорение	Ac	$м/с^2$	Абсолютное значение ускорения, испытываемое аппаратом
Ускорение от двигателей	Ae	$м/с^2$	Абсолютное значение ускорения, создаваемого включенным двигателем
Ускорение силы Стокса	As	$м/с^2$	Абсолютное значение ускорения, создаваемого силой аэродинамического сопротивления

Расширенная телеметрия включает также сообщения обо всех изменениях в работе аппарата. Устройство расширенной диагностики позволяет вам также отправлять специальные текстовые сообщения на Землю с используя программный вызов `telemetry_send_message` в коде программы полета. Расширенная телеметрия в процессе полета выглядит так:

```
F=050.0 Devs=C1+;G1+;Dm1+;T1+;DA1+;CPU1+;EG1-;FT1+;Pam1+;Hsl1D
```

Возможные параметры расширенной телеметрии представлены в таблице:

Название	Код	Единица измерения	Комментарий
Масса доступного топлива	F	кг	
Состояния устройств	Devs	-	Возможные состояния: **+** включено; **-** выключено; **D** не доступно.

РЕШЕНИЕ:

Пример программы полета, аккуратно реализующей данный алгоритм для стартовой высоты 80000 м:

```
1. hs = True
2. engine = False
3. probe.set_device_period('D1', 10)
4. probe.set_device_period('DA1', 10)
5. while probe.run():
6.     t = probe.cpu_get_flight_time()
7.     if hs and t >= 236:
8.         probe.set_device_state('Hsl1', STATE_OFF)
9.         probe.set_device_state('Pam1', STATE_ON)
```

```
10.         hs = False
11.         continue
12.     if not engine and 323 <= t < 336:
13.         probe.set_device_state('EL1', STATE_ON)
14.         engine = True
15.         continue
16.     if engine and t >= 336:
17.         probe.set_device_state('EL1', STATE_OFF)
18.         engine = False
19.         continue
20.     if probe.navigation_has_landed():
21.         break
```

Задача 2.3. «Работа на поверхности Марса» (макс. 20000 баллов)

2.3.1. Конструирование аппарата

Мы предлагаем вам продолжить миссию по покорению Марса разработкой аппарата для работы на поверхности планеты. Совершив удачную посадку, ваш аппарат начнет передавать на Землю научные данные, которые позволят углубить знания человечества о Красной планете.

Условия победы:

За успешное решение этой миссии вы получаете победные баллы. В этой миссии вы получаете баллы за переданную на Землю научную информацию следующим образом:

За каждый переданный на землю 1 Мегабит (1000 килобит) научной информации команда получает 0,1 балла (другими словами, количество баллов = переданная информация / 1000).

Вам дается **10 попыток**. Каждая дополнительная попытка сверх этих десяти отнимает у команды 30 баллов. Таким образом можно получить не более **30** дополнительных попыток (т. е. Минус 900 баллов).

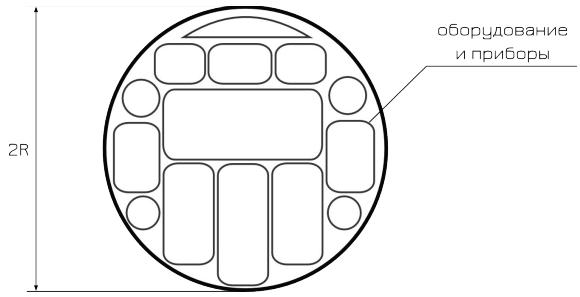
В миссии также возможны следующие достижения:

- **Исследователь** — Передать научные данные с Марса любого объема (100 балл.)
- **Первопроходец** — Первыми по времени передать научные данные с Марса (200 балл.)
- **Доставка оборудования** — Самый тяжелый аппарат, севший на Марс и передавший данные (300 балл.)
- **Экономичность** — Самый легкий аппарат, севший на Марс и передавший данные (500 балл.)
- **Космическая гонка** — Передать научные данные с Марса в течение первых трех суток с момента получения задачи командой (300 балл.)

- **Надежная конструкция** — Аппарат проработал на Марсе все 72 часа (200 балл.)

2.3.2. Постановка задачи

Если с точки зрения физики задача остается прежней, то конструкторская ее часть будет заметно сложнее. Вам предстоит полностью сконструировать аппарат и составить программу не только посадки, но и планетарной активности. Аппарат может проработать на поверхности Марса не более **72 земных (!) часов.**



Вам не придется конструировать произвольный аппарат с нуля. В вашем распоряжении будет аппарат сферической формы, его размер вы можете установить самостоятельно. Вам придется рассчитать требуемые внешние параметры аппарата (массу и радиус), выбрать необходимое для работы оборудование и научные приборы.

Мы рекомендуем следующий порядок разработки:

1. сконструировать аппарат с максимальной полезной нагрузкой и посадить его;
2. наполнить полезную нагрузку необходимым оборудованием для обеспечения максимальных научных результатов.

Конструкция аппарата и программа полета должны учитывать уровень энергоснабжения в корабле, чтобы всем системам хватало энергии, а также пропускную способность систем связи, чтобы информация могла быть передана на Землю в нужном объеме.

Помимо конструкции аппарата вам предстоит разработать программу полета, например, определить время, когда должны включаться и выключаться тормозные двигатели или научные приборы.

2.3.3. Общие параметры аппарата

Параметры, связанные с посадкой на Марс, можно взять из предыдущей задачи. В таблице указаны общие параметры, связанные с конструированием аппарата:

Параметр	Пояснение	Значение
m_{max}	Максимальная общая масса аппарата на начало полета	20 000 кг (20 т)
m	Общая масса аппарата	вычисляется по формуле:

		$m = m_c + \sum_i m_{d(i)}$, где $\mathbf{m_c}$ — это масса конструкции аппарата, а $\mathbf{m_{d(i)}}$ — масса каждого из установленных устройств
m_c	Масса конструкции аппарата	вычисляется по формуле $m_c = V \cdot \rho_c$, где $\boldsymbol{\rho_c}$ — средняя плотность конструкции аппарата, а $\mathbf{V}$ — объем аппарата
ρ_c	Средняя плотность конструкции аппарата	100 кг/м^3
V	Объем сферического аппарата	вычисляется по формуле: $V = \frac{4}{3}\pi r^3$, где **r** — радиус аппарата; суммарный объем всех устройств **не должен превышать** объема аппарата
r_{max}	Максимальный радиус аппарата	2 м
a_{max}	Предельная перегрузка аппарата	155 м/с^2

Далее перечисляются устройства, из которого может быть скомпонован аппарат.

Важно отметить, что при раскрытии парашюта в атмосфере аппарат может превысить предельную перегрузку (см. в таблице выше). В этом случае аппарат разрушается перегрузками.

2.3.4. Устройства для посадки аппарата

В таблице представлены устройства, которые могут использоваться при посадке:

Название	Код	Масса, кг	Объем, м^3	Скорость, м/с	Параметр
Двигатель тормозной	E	1900	0,7	800	Расход топлива Δm = 0,5 кг/с
Двигатель посадочный	EG	250	1,0	3600	Расход топлива Δm = 4,2 кг/с
Двигатель посадочный легкий	EL	100	0,5	3600	Расход топлива Δm = 2 кг/с
Двигатель маневровый	EM	140	0,025	800	Расход топлива Δm = 0,02 кг/с
Топливный бак базовый	FT	50	0,04	-	Топливо: 50 кг
Топливный бак большой	FTL	500	0,4	-	Топливо: 500 кг
Парашют	Pa	40	0,02	35	Площадь: 10 м^2
Парашют для разреж. атмосферы	Pam	10	0,06	600	Площадь: 200 м^2
Силовой парашют	Fp	70	0,02	500	Площадь: 2 м^2

Аэродинамический тормоз	Hsl	150	0,01	7000	Площадь: 18 м^2
Усиленный аэродинамический тормоз	Hs	1000	0.01	4000	Площадь: 3 м^2
Демпфер	Dm	400	0,08	40	-
Демпфер с амортизирующими опорами	DAM	1000	0,15	50	-

При выборе аэродинамического тормоза для посадки важно учитывать ограничения:

- аппарат может содержать только одно устройство типа аэродинамический тормоз;
- аэродинамический тормоз должен быть включен с самого начала полета;
- напоминаем, что **одновременное открытие** двух парашютов и/или парашюта и аэродинамического тормоза приведет к отбрасыванию обоих устройств.

При этом несколько двигателей могут работать одновременно.

2.3.5. Основные устройства и устройства связи

Следующие устройства необходимы для нормальной работы аппарата и связи с Землей (устройства электропитания см. в отдельном документе):

Название	Код	Масса, кг	Объем, м^3
Центральный компьютер: необходим на каждом аппарате, чтобы он функционировал	CPU	50	0,04
Генератор радиоизотопный	G	200	0,04
Аккумулятор	Acm	70	0,02
Солнечная батарея	Sbt	30	0,06
Солнечная батарея расширенная	Sbe	80	0,14
Диагностика базовая	D	25	0,01
Диагностика расширенная	DA	80	0,02
Передатчик базовый	T	30	0,01
Передатчик широкополосный	WT	160	0,05

2.3.6. Получение научных данных на планете

Итогом работы любого аппарата является передача на Землю данных, производимых научными приборами. На аппарат можно установить несколько приборов одного типа — если будет достаточно питания и пропускной способности канала, они все будут передавать собранную научную информацию. При этом научная информация от приборов одного типа учитывается только до момента достижения соответствующего предела (информация этого типа становится на Земле более не интересной).

В следующей таблице представлены все научные приборы:

Название	Код	Масса, кг	Объем, $м^3$	Предел научной информации, Кбит
Барометр	Brm	8	0,003	1 000 000
Видеокамера	VC	20	0,008	800 000
Газовый хроматограф	GCh	105	0,06	5 500 000
Датчик магнитного поля	Mgm	10	0,003	1 500 000
Инфракрасная камера	IRC	10	0,004	3 000 000
Камера	C	10	0,005	700 000
Лазерный анализатор свойств атмосферы	LID	15	0,005	2 000 000
Лазерный испаритель образцов	LEv	20	0,025	1 000 000
Масс-Спектрометр	MSS	200	0,08	7 000 000
Радиометр	Rm	14	0,02	1 200 000
Сейсмограф	Smg	55	0,08	3 500 000
Спектрометр	S	80	0,04	4 600 000
Термометр	Trm	2	0,003	500 000

Чем больше вы передадите на Землю научной информации, тем большее количество победных балл. вы получите.

2.3.7. Электрическая подсистема аппарата

Разработка аппарата для посадки на Марс включает в себя создание электрической подсистемы: необходимо обеспечить бесперебойное питание всех устройств, необходимых в процессе посадки и при работе на поверхности.

Каждое устройство имеет следующую характеристику: **уровень потребления** или **генерации** энергии.

2.3.7. Производство электроэнергии

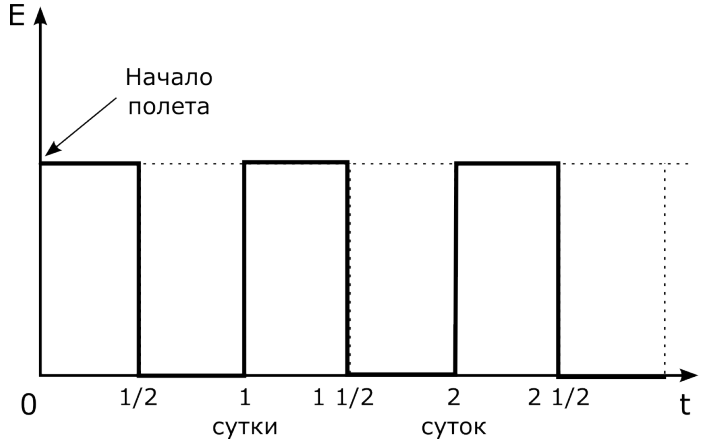
В вашем распоряжении несколько устройств, способного производить энергию (часть из них можно использовать только на поверхности):

Название	Код	Производство энергии, 10 * Вт
Генератор радиоизотопный	G	50

Аккумулятор	Acm	0 (емкость 400 * 10Вт/ч)
Солнечная батарея	Sbt	10
Солнечная батарея расширенная	Sbe	32

Генератор производит энергию, когда **включен**. Мы допускаем, что мощность его не меняется в течение времени. Аккумулятор позволяет компенсировать недостаток энергии в сети, пока его емкость не стала равна 0. Если в сети существуют излишки энергии, аккумулятор подзаряжается. Считается, что емкость аккумулятора остается постоянной.

2.3.8. Солнечные батареи



Солнечные батареи генерируют энергию **только после посадки** аппарата и только днем. Время суток на Марсе — 24 ч 39 мин 35 сек, день считается за первую половину этого периода. Первые сутки начинаются с начала полета.

Вся производимая энергия складывается. Можно рассматривать все множество производящих и потребляющих энергию устройств как единый пул.

2.3.9. Потребление электроэнергии

Устройства аппарата потребляют энергию только когда они **включены**. Все устройства объединены в общую сеть: потребление всех приборов суммируется, как и производство электроэнергии. Если в какой-то момент потребление устройств превышает производимую электроэнергию, аппарат переходит в т. н. **«экономичный режим»**. В таблице представлены все устройства, которые потребляют электроэнергию:

Название	Тип	Код	Потребление энергии, 10 * Вт
Центральный процессор	Центр. процессор	CPU	10
Двигатель тормозной	Двигатель	E	25
Двигатель посадочный	Двигатель	EG	5
Двигатель посадочный легкий	Двигатель	EL	5
Двигатель маневровый	Двигатель	EM	11

Название	Тип	Код	Потребление энергии, 10 * Вт
Диагностика Базовая	Диагностика	D	2
Диагностика Расширенная	Диагностика	DA	10
Барометр	Научный прибор	Brm	2
Видеокамера	Научный прибор	VC	24
Газовый хроматограф	Научный прибор	GCh	9
Датчик магнитного поля	Научный прибор	Mgm	9
Инфракрасная камера	Научный прибор	IRC	16
Камера	Научный прибор	C	14
Лазерный анализатор свойств атмосферы	Научный прибор	LID	6
Лазерный испаритель образцов	Научный прибор	LEv	80
Масс-Спектрометр	Научный прибор	MSS	19
Радиометр	Научный прибор	Rm	8
Сейсмограф	Научный прибор	Smg	2
Спектрометр	Научный прибор	S	28
Термометр	Научный прибор	Trm	2
Передатчик Базовый	Передатчик	T	4
Передатчик Широкополосный	Передатчик	WT	8

2.3.10. Экономичный режим

При переходе аппарата в экономичный режим (SAFE MODE), остаются включенными только те устройства, которые критически необходимы для работы аппарата. При создании аппарата вы можете для каждого из устройств указать его состояние в экономичном режиме, выбрав из вариантов:

- **ON** — устройство будет включено при входе аппарата в экономичный режим;
- **OFF** — устройство будет выключено при входе аппарата в экономичный режим;
- **не указывать состояние** — устройство продолжит работать в том состоянии, в котором находилось на момент входа аппарата в экономичный режим.

Выйти из экономичного режима **нельзя**. Если в какой-то момент работы экономичного режима потребление устройств превышает производимую электроэнергию, аппарат **выключается**, связь с ним прерывается.

2.3.11. Подсистема связи

Разработка аппарата для посадки на Марс включает в себя разработку подсистемы связи: необходимо обеспечить передачу ценной информации с аппарата на Землю в процессе посадки и при работе на поверхности.

Все устройства, которые передают информацию, характеризуются следующими параметрами:

- **генерация трафика / ширина канала** — объем данных в секунду (в Кбит), который производит / передает данное устройство;
- **минимальный** и **максимальный период** передачи — возможность задать для устройства период передачи, что позволяет осуществлять передачу не каждую секунду, а с заданным интервалом. Минимальный период передачи для всех устройств — 1 секунда.

2.3.12. Устройства связи

В вашем распоряжении будут устройства, которые позволяют передавать данные на землю. В таблице представлены доступные варианты устройств связи:

Название	Код	Ширина канала, Кбит/с
Передатчик Базовый	T	20
Передатчик Широкополосный	WT	100

Передатчики отправляют информацию на Землю, когда **включены**.

Ширина каналов для всех работающих передатчиков складывается. Можно рассматривать все множество производящих и передающих трафик устройств как единый пул.

Важно учесть: если все устройства аппарата производят больше данных в секунду, чем может быть передано доступными устройствами связи, не все данные будут переданы на Землю. Поскольку нет возможности задать приоритет передаваемой информации, вы не сможете определить заранее, какая информация не попадет в канал.

2.3.13. Устройства диагностики

Устройства диагностики — это особенные устройства, которые передают на Землю телеметрию — информацию о системах аппарата. Чтобы отправить данные, устройство диагностики должно быть включено, а ширина канала должна позволять передачу телеметрии. Вы можете **установить период передачи**, тем самым снизив нагрузку на канал,

о чем рассказывается далее.

Название	Код	Период передачи мин/макс, с	Генерация трафика, Кбит/с
Диагностика Базовая	D	1/3600	2
Диагностика Расширенная	DA	1/3600	2

2.3.14. Научное оборудование

Как правило, основной трафик генерируется научными приборами. Каждый научный прибор может генерировать определенный поток данных, который может быть отправлен на Землю через устройства связи. В таблице представлены доступные научные приборы:

Название	Код	Период передачи мин/макс, с	Генерация трафика, Кбит/с	Предел научной информации, Кбит
Барометр	Brm	1/10	3	1 000 000
Видеокамера	VC	1/10	26	800 000
Газовый хроматограф	GCh	1/10	25	5 500 000
Датчик магнитного поля	Mgm	1/10	9	1 500 000
Инфракрасная камера	IRC	1/100	15	3 000 000
Камера	C	1/100	18	700 000
Лазерный анализатор свойств атмосферы	LID	1/10	8	2 000 000
Лазерный испаритель образцов	LEv	1/10	98	1 000 000
Масс-Спектрометр	MSS	1/100	22	7 000 000
Радиометр	Rm	1/100	10	1 200 000
Сейсмограф	Smg	1/10	11	3 500 000
Спектрометр	S	1/100	50	4 600 000
Термометр	Trm	1/10	2	500 000

Вы можете также **установить период передачи**, чтобы снизить нагрузку на канал. Каждый прибор характеризуется пределом научной информации — данные этого типа становятся не интересны на Земле (и не учитываются при определении победителя), даже если вы передали больше этого предела.

2.3.15 Установка периода передачи

Для всех устройств, генерирующих информацию, Вы можете **установить период**

передачи с помощью специальной команды `set_period` в программах полета или работы на поверхности, тем самым пропорционально снизить нагрузку на канал. Так, при установке периода передачи Базовой Диагностики в 10 секунд, вы будете получать данные телеметрии только раз в 10 секунд, но нагрузка на канал снизится до 0,2 Кбит/с. **Важно:** данные передаются **пакетно**, то есть при отсутствии доступного канала шириной 0,2 Кбит/с, телеметрия не будет передана.

2.3.16. Анализ телеметрии

После запуска аппарата, вам доступны записи переданной телеметрии. **Важно:** телеметрия аппарата будет получена на Земле, если аппарат обладает **устройством диагностики и передатчиком** (эти устройства должны быть включены, на них должно хватать **энергии**).

Есть два типа телеметрии: **базовая** и **расширенная**. **Важно:** устройство расширенной диагностики не обеспечивает базовой телеметрии! Для получения телеметрии каждого типа необходимо установить на аппарате соответствующее оборудование и включить его. Параметры телеметрии **во время полета** были описаны в задаче о посадке на Марс.

Базовая телеметрия после посадки выглядит так:

```
Ti=00:01:30 PB=058 BW=10.1/20.0
```

Возможные параметры телеметрии представлены в таблице:

Время полета	Ti	час:мин:сек	Период телеметрии может быть изменен командой PERIOD устройству диагностики
Баланс энергии	PB	10 Вт	Суммарное производство энергии минус суммарное ее потребление устройствами аппарата
Канал связи	BW	Кбит	Используемая / Доступная ширина канала связи

Расширенная телеметрия после посадки выглядит так:

```
PG=50 PC=22 BWG=20 BWC=2 SI=0000010.0
Devs=C1+;G1+;Dm1+;T1+;DA1+;CPU1+;EG1-;FT1-;Pam1D;Hsl1D
```

Вам доступна информация о генерации и потреблении электроэнергии и трафика, состояния устройств, а также следующая информация:

Научная информация	SI	Кбит	Объем переданной информации

РЕШЕНИЕ:

Аппарат со следующей конструкцией и программа полета удовлетворяют решению

задачи для начальной высоты 80 км.

Радиус сферического аппарата: 0.81 м

Выбранные устройства (в скобках указано начальное состояние):

- Аэродинамический тормоз (ON)
- Генератор радиоизотопный (ON)
- Двигатель посадочный (OFF)
- Демпфер (ON)
- Диагностика базовая (ON)
- Камера (OFF)
- Парашют для разреженной атмосферы (OFF)
- Передатчик базовый (ON)
- Топливный бак базовый (ON)
- Центральный процессор (ON)

Программа полета:

```
01. hs = True
02. engine = False
03. landed = False
04. probe.set_device_period('D1', 10)
05. while probe.run():
06.     if not landed:
07.         if hs and probe.cpu_get_flight_time() >= 245:
08.             probe.set_device_state('Hsl1', STATE_OFF)
09.             probe.set_device_state('Pam1', STATE_ON)
10.             hs = False
11.             continue
12.         if not engine and probe.cpu_get_flight_time() >= 283:
13.             probe.set_device_state('EG1', STATE_ON)
14.             engine = True
15.             continue
16.         if not landed and probe.navigation_has_landed():
17.             landed = True
18.             probe.set_device_state('C1', STATE_ON)
```

Задача 2.4. «Смотрим на Землю» (макс. 160 баллов)

Тренировочная миссия «Смотрим на Землю» призвана познакомить участников с возможностями симулятора при решении второй части задач и, конкретно, с задачей ориентации космического аппарата (КА) на орбите Земли.

2.4.1. Условия победы

За успешное решение этой миссии вы получаете победные баллы. В миссии возможны следующие достижения:

Первые на орбите — Выполнить миссию первыми (60 балл.)

Вторые на орбите — Выполнить миссию вторыми (50 балл.)

Третьи на орбите — Выполнить миссию третьими (40 балл.)

С первой попытки — Выполнить миссию с первой попытки (50 балл.)

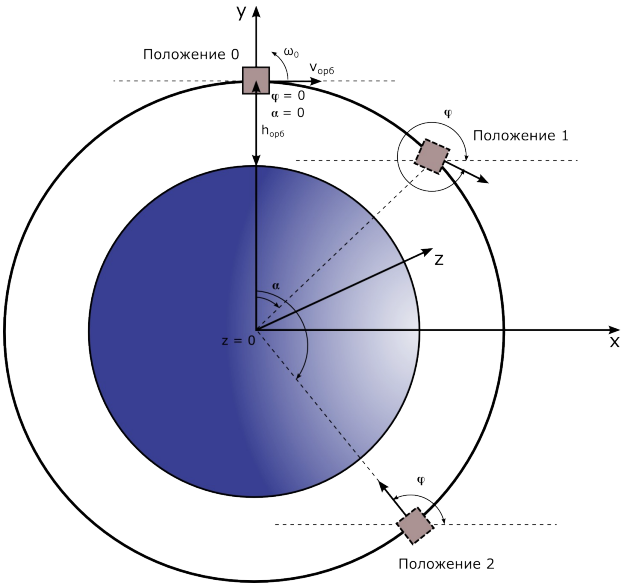
Со второй попытки — Выполнить миссию со второй попытки (30 балл.)

Миссия выполнена — Выполнить миссию с третьей и более попытки (10 балл.)

Выполнил на один виток — Аппарат выполнил миссию за единственный виток вокруг Земли (50 балл.)

2.4.2. Постановка задачи

Для решения многих реальных задач бывает необходимо сориентировать КА **в надир** (**нормально** по отношению к поверхности), что соответствует Положению 2 на схеме.



КА движется по круговой орбите с заданной высотой в плоскости X0Y. Положение КА в любой момент времени задается углом α, который увеличивается по часовой стрелке. КА имеет также угол ориентации φ, который увеличивается против часовой стрелки.

КА ориентирован в надир (нормально по отношению к поверхности), если выполняется соотношение: $\alpha + \varphi = 270°$.

Необходимо запрограммировать аппарат так, чтобы он погасил начальную угловую скорость ω_0, сориентировался нормально по отношению к Земле и совершил один оборот вокруг Земли, оставаясь все это время сориентированным в надир.

Анализ телеметрии после неудачного запуска позволит исправить ошибки, допущенные при расчете или программе полета КА.

В данной миссии вам не потребуется конструировать аппарат. Даже программа полета будет частично написана за вас. Поскольку задача может быть решена аналитически, необходимо будет рассчитать параметры констант, которые используются в программе полета КА (см. далее).

КА оснащен подсистемой ориентации и стабилизации, которая отслеживает угол ориентации и угловую скорость, а также может задавать КА момент вращения посредством включения маховика. В программе полета можно задать момент вращения, который не будет превышать предельных характеристик подсистемы. В ходе решения этой задачи мы будем считать, что масса аппарата остается неизменной, а форма — идеальный куб с известной длиной грани.

2.4.3. Исходные данные

Параметр	Пояснение	Значение
G	Гравитационная постоянная	$6{,}6742 \cdot 10^{-11}$ Н м^2/кг^2
M	Масса Земли	$5{,}9726 \cdot 10^{24}$ кг
R	Радиус Земли	6 371 032 м
$h_{орб}$	Высота стартовой орбиты	См. уникальные условия миссии, м
m	Масса КА (не изменяется)	2,4 кг
ω_0	Начальная угловая скорость	1 °/с
$\mathbf{v_{обр}}$	Стартовая орбитальная скорость	Вычисляется по формуле: $v_{орб} = \sqrt{\frac{GM}{R + h_{орб}}}$, м/с
T	Период обращения аппарата вокруг Земли	Вычисляется по формуле: $T = 2\pi \frac{R + h_{орб}}{v_{орб}}$, с
$\omega_З$	Угловая скорость движения аппарата вокруг Земли	Вычисляется по формуле: $\omega_З = \frac{360°}{T}$, °/с
$M_{макс}$	Предельный момент маховика СУОС	0,000023 Н м
I	Момент инерции кубического КА (в предположении, что масса распределена равномерно по его объему)	Вычисляется по формуле: $I_Z = \frac{1}{12}(2a^2)m$, кг м^2
a	Сторона грани кубического аппарата	0,1032 м
ε	Угловое ускорение КА	Вычисляется по формуле: $\varepsilon = \frac{M}{I_Z}$, °/с^2 где M — момент, с которым работает маховик.

КА содержит следующую программу полета на языке Python:

```
t = # ВРЕМЯ РАБОТЫ МАХОВИКА
w = # КОНЕЧНАЯ УГЛОВАЯ СКОРОСТЬ
M0 = # МОМЕНТ
M = 0.000001
dw = 0.01

mode = 'rotate'
sputnik.orientation.set_motor_moment(AXIS_Z, M0);
sputnik.orientation.start_motor(AXIS_Z);
moment = True
```

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

    if mode == 'rotate' and sputnik.cpu.get_flight_time() >= t:
        mode = 'ok'
        sputnik.orientation.stop_motor(AXIS_Z)
        moment = False

    if mode == 'ok':
        av = sputnik.orientation.get_angular_velocity(AXIS_Z)
        if abs(av - w) < dw:
            if moment:
                sputnik.orientation.stop_motor(AXIS_Z)
                moment = False
        elif not moment:
            sputnik.orientation.start_motor(AXIS_Z)
            moment = True
            if av > w:
                sputnik.orientation.set_motor_moment(AXIS_Z, -M)
            else:
                sputnik.orientation.set_motor_moment(AXIS_Z, M)
```

Данная программа содержит четыре части:

- объявление глобальных констант и переменных;
- запуск маховика в самом начале полета;
- остановку маховика при достижении определенного времени и переход к стабилизации полета;
- стабилизация ориентации в надир через сохранение необходимой угловой скорости.

Эта программа не является законченной. Необходимо рассчитать и задать значения трем константам:

Переменная	Пояснение
w	Угловая скорость, которую аппарат должен иметь на финальном витке облета (ω, °/с)
t	Время работы маховика или цикла 1 (t, с)
M0	Момент, который нужно сообщить маховику в начале полета (M_0, Н м)

Значения этих констант могут быть получены путем решения следующей системы уравнений:

$$\begin{cases} \alpha+\varphi=270° \\ \alpha=\omega_3 t \\ \varphi=\omega_0 t+\dfrac{\varepsilon t^2}{2} \\ \omega=\omega_0+\varepsilon t \\ \omega=-\omega_3 \\ M_0=I_z\varepsilon \end{cases}$$

Угловая скорость, которую аппарат приобретает к финальному витку, должна быть по

модулю равна угловой скорости движения КА по орбите (знак минус связан с тем, что углы α и φ направлены в разные стороны).

Система уравнений имеет следующее решение:

$$\omega = \frac{-360° \cdot \sqrt{\frac{GM}{R+h_{орб}}}}{2\pi(R+h_{орб})}$$

$$t = \frac{2 \cdot 270°}{\omega_0 - \omega}$$

$$M_0 = \frac{(\omega - \omega_0) \cdot I_z}{t}$$

ВАЖНО: для того, чтобы данная программа сработала верно, необходимо максимально точно рассчитать значения этих трех переменных и ввести их в программу полета с максимальным числом знаков после запятой. Рекомендуется использовать не менее 4 значащих цифр после запятой.

Также вы можете написать собственную более универсальную программу полета, которая будет достигать стабилизации аппарата без предварительных расчетов.

Для решения данной миссии мы рекомендуем вам обратиться к разделу 2.2 «Механический расчет» и разделу 3.1 «Телеметрия полета» большого описания модели «Орбита» (http://orbitagame.ru/nti-2/media/pdf/orbita2.pdf), а также «Руководству по программированию аппарата на языке Python для управления подсистемами аппарата» (http://orbitagame.ru/nti-2/media/pdf/python-programming-2.pdf).

РЕШЕНИЕ:

Для решения этой задачи достаточно аккуратно рассчитать значения трех переменных для заданного варианта и вставить их в код программы полета:

```
t = 508.427177612
w = -0.0620990060685
M0 = -8.899295566e-06
M = 0.000001
dw = 0.00001

mode = 'rotate'
sputnik.orientation.set_motor_moment(AXIS_Z, M0);
sputnik.orientation.start_motor(AXIS_Z);
moment = True

while sputnik.cpu.run():

    if mode == 'rotate' and sputnik.cpu.get_flight_time() >= t:
        mode = 'ok'
        sputnik.orientation.stop_motor(AXIS_Z)
        moment = False

    if mode == 'ok':
```

```
        av = sputnik.orientation.get_angular_velocity(AXIS_Z)
        if abs(av - w) < dw:
            if moment:
                sputnik.orientation.stop_motor(AXIS_Z)
                moment = False
        elif not moment:
            sputnik.orientation.start_motor(AXIS_Z)
            moment = True
            if av > w:
                sputnik.orientation.set_motor_moment(AXIS_Z, -M)
            else:
                sputnik.orientation.set_motor_moment(AXIS_Z, M)
```

Задача 2.5 «Связь с Землей» (макс. 520)

Следующая тренировочная миссия «Связь с Землей» знакомит участников с передачей сообщений с КА на Землю через подсистему высокопроизводительной связи.

2.5.1. Условия победы

За успешное решение этой миссии вы получаете победные баллы. В миссии возможны следующие достижения:

Первые на орбите — Выполнить миссию первыми (200 балл.)

Вторые на орбите — Выполнить миссию вторыми (175 балл.)

Третьи на орбите — Выполнить миссию третьими (150 балл.)

С первой попытки — Выполнить миссию с первой попытки (120 балл.)

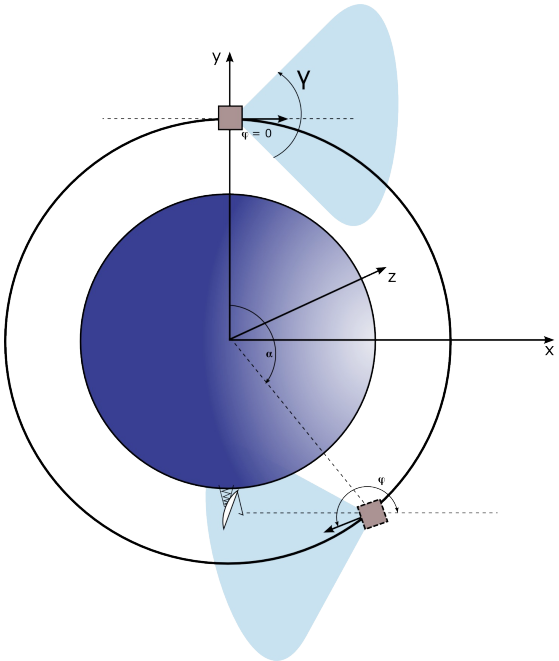
Со 2-3 попытки — Выполнить миссию со второй или третьей попытки (100 балл.)

Миссия выполнена — Выполнить миссию с третьей и более попытки (80 балл.)

Выполнил на один виток — Аппарат выполнил миссию за единственный виток вокруг Земли (200 балл.)

2.5.2. Постановка задачи

Как и в предыдущей миссии КА движется по круговой орбите с заданной высотой в плоскости X0Y. Необходимо запрограммировать аппарат так, чтобы он передал на Землю заданное сообщение. При этом необходимо воспользоваться высокопроизводительной связью КА. Задача усложняется двумя факторами: сигнал экранируется Землей, антенна такой подсистемы имеет угол раскрыва (γ), заданный в параметрах КА.



Мы будем считать, что наземный измерительный пункт (НИП) отслеживает положение КА, поэтому потребуется только сориентировать аппарат на НИП.

В данной миссии вам не потребуется конструировать аппарат целиком, однако нужно

будет подобрать несколько параметров конструкции аппарата — площади солнечных батарей и радиаторов, а также написать программу полета. Мы рекомендуем вам использовать наработки, полученные в предыдущей миссии.

КА оснащен подсистемой ориентации и стабилизации, которая позволяет задавать момент вращения посредством включения маховика, а также подсистемой высокопроизводительной связи, параметры которой указаны в таблице ниже. КА как и в первой тренировочной миссии в начале полета будет иметь стартовую угловую скорость, которую придется погасить для успешного выполнения миссии.

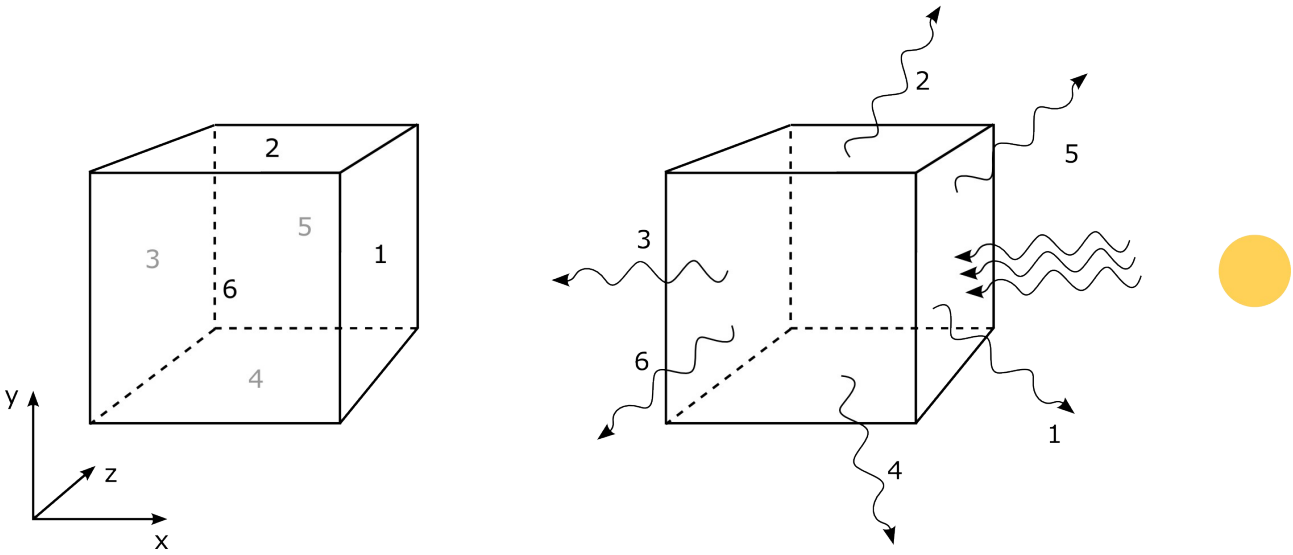
2.5.3. Исходные данные

Параметр	Пояснение	Значение
$h_{орб}$	Высота стартовой орбиты	См. уникальные условия миссии, м
m	Масса КА (не изменяется)	5,5 кг
ω_0	Начальная угловая скорость	1 °/с
$M_{макс}$	Предельный момент маховика СУОС	0,0026 Н м
a	Сторона грани кубического аппарата	0,15037 м
$P^1_{потр}$ $Q^1_{внутр}$	Потребление электроэнергии аппаратом при выключенной подсистеме высокопроизв. связи	8,8 Вт
$P^2_{потр}$ $Q^2_{внутр}$	Потребление электроэнергии аппаратом при **включенной** подсистеме высокопроизв. связи	9,8 Вт
$\eta_{ФЭП}$	КПД солнечных батарей	29,8%
$S_{ФЭП}$	Площадь солнечных батарей	Вычисляется по формуле: $S_{ФЭП} = k_{ФЭП} a^2$, м^2
$k_{ФЭП}$	Доля солнечных батарей на освещенной грани аппарата (1-4)	Параметр конструирования (см. далее)
q_C	Плотность потока солнечного излучения	1400 на солнечной стороне, 0 на теневой стороне
$P_{аккум}$	Емкость аккумулятора КА	41,8 Вт ч
$A_{сб}$	Коэффициент поглощения солнечными батареями	0,95
$A_{рад}$	Коэффициент поглощения радиаторами	0,2
$\varepsilon_{сб}$	Степень черноты солнечных батарей	0,4
$\varepsilon_{сб}$	Степень черноты радиатора	1
γ	Угол раскрыва антенны высокопроизводительной связи	180°

T_0	Начальная температура аппарата	290 К
$T_{мин}$	Минимально допустимая температура для КА	263 К
$T_{макс}$	Максимально допустимая температура для КА	313 К
c	Средняя теплоемкость КА	800 Дж / (кг К)
σ	Постоянная в законе Стефана-Больцмана	$5{,}67 \cdot 10^{-8}$ Дж·с$^{-1}$·м$^{-2}$ · К$^{-4}$

2.5.4. Конструирование аппарата

Аппарат имеет кубическую форму. Каждая из граней аппарата имеет свой номер:



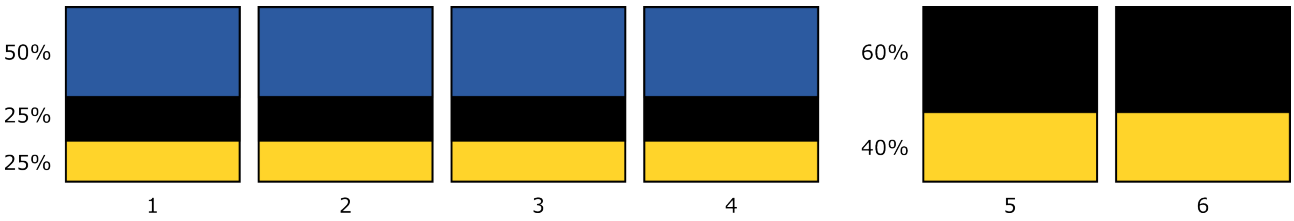
В начале полета аппарат ориентирован, как показано на левом рисунке. В процессе полета аппарат может вращаться вокруг оси Z, так что грани **1-4** могут последовательно освещаться Солнцем, которое всегда находится в положительной бесконечности на оси X. При этом грани 5-6 всегда остаются в тени.

Это важно при проектировании энергетической подсистемы и системы обеспечения теплового режима:

На гранях 1-4 могут быть расположены солнечные батареи. Считается, что при нахождении аппарата на солнечной стороне орбиты, одна из его граней полностью освещается солнцем, а все другие находятся в тени. Энергия, получаемся от солнечных батарей может быть рассчитана по формуле, указанной выше.

Освещаемая солнцем грань нагревается солнечными лучами. Одновременно с этим все грани излучают тепло в комическое пространство. Теплообмен осуществляется через солнечные батареи и радиаторы. Площадь радиаторов указывается отдельно для граней 1-4 и 5-6. Оставшиеся площади граней покрываются специальной защитной пленкой, которая препятствует теплообмену.

При конструировании аппарата вам необходимо рассчитать и указать площади для солнечных батарей и радиаторов на гранях 1-4 аппарата и площади радиаторов на гранях 5-6 аппарата, например:



При этом не учитываются масса солнечных батарей и радиаторов — ими можно принебречь.

2.5.5. Расчет подсистемы электропитания

Подсистема электроэнергии состоит из аккумуляторов и солнечных батарей и должна обеспечивать работу всех остальных подсистем аппарата, общее потребление которых равно $P_{потр}$ (отличается в зависимости от того, включена ли подсистема высокопроизводительной связи). При недостаточной генерации электроэнергии солнечными батареями начинает расходоваться запасенная в аккумуляторе энергия. Если же в сети существуют излишки энергии, аккумулятор заряжается до своей максимальной емкости. Максимальную емкость аккумулятора считаем постоянной. В начале полета емкость аккумулятора равна максимальной, что составляет $P_{аккум}$.

Солнечные батареи вырабатывают энергию, которая вычисляется по формуле:

$P_{ФЭП}(t) = \eta_{ФЭП} S_{ФЭП} q_c(t)$, Вт.

Таким образом, необходимо вычислить и задать площадь солнечных батарей $S_{ФЭП}$ (на гранях 1-4) так, чтобы вырабатываемой ими энергии хватало не только на поддержание работы аппарата на солнечной стороне, но и на подзарядку аккумулятора после возвращения аппарата с теневой стороны орбиты.

2.5.6. Расчет подсистемы обеспечения теплового режима

Считается, что аппарат имеет единую температуру во всем внутреннем объеме. В начале полета температура аппарата равна T_0. Каждая из подсистем имеет рабочий диапазон температур, в таблице выше приведены минимальная $T_{мин}$ и максимальная $T_{макс}$ температура аппарата, при которой все подсистемы будут функционировать нормально.

Изменение температуры аппарата происходит в связи с внешним теплообменом (поглощением и излучением тепла аппаратом), а также внутренним нагревом аппарата в связи с работой подсистем. Изменение температуры задается уравнением:

$$c \cdot m(t) \cdot \frac{\Delta T}{\Delta t} = Q^{\Sigma}_{внеш}(t) + Q^{\Sigma}_{внутр}(t)$$

Тепло, выделяемое подсистемами аппарата равно потребляемой электроэнергии (см. в таблице с исходными данными, зависит от того, какие устройства включены). Внешний теплообмен задается формулой:

$$Q^{\Sigma}_{внеш}(t) = \left(S_{сб}^{погл} A_{сб} + S_{рад}^{погл} A_{рад} \right) q_c(t) - \left(S_{сб}^{изл} \varepsilon_{сб} + S_{рад}^{изл} \varepsilon_{рад} \right) \sigma T^4 ,$$

где $S_{сб}^{погл}$ – площадь поглощения тепла солнечными батареями (солнечная батарея на одной из граней 1-4), м^2; $A_{сб}$ – коэффициент поглощения солнечного излучения солнечными батареями; $S_{рад}^{погл}$ – площадь поглощения тепла от солнца радиаторами (радиатор на одной из граней 1-4), м^2; $A_{рад}$ – коэффициент поглощения солнечного излучения радиаторами; $S_{сб}^{изл}$ – площадь излучения всеми солнечными батареями КА (солнечные батареи на всех гранях 1-4), м^2; $\varepsilon_{сб}$ – степень черноты солнечных батарей; $S_{рад}^{изл}$ – площадь излучения всеми радиаторами КА (радиаторы на гранях 1-4 и 5-6), м^2, $\varepsilon_{рад}$ – степень черноты радиатора.

Необходимо рассчитать площадь радиаторов так, чтобы при смене солнечной и теневой части орбиты, температура аппарата сохранялась в требуемом диапазоне.

2.5.7. Отправка сообщения на Землю

Написание программы полета потребует обращения к подсистеме transmitter, которая представляет в программе подсистему высокопроизводительной радиосвязи. Прежде чем отправлять и принимать сообщения по высокопроизводительной радиосвязи, необходимо включить соответствующую подсистему (т. к. в начале полета она выключена), сделать это можно, изменив ее режим на «STATE_ON»:

```
sputnik.transmitter.set_state(STATE_ON)
```

Важно учесть, что при включении этой подсистемы хоть и незначительно, увеличивается расход электроэнергии и тепло, выделяемое внутри аппарата.

Для отправки сообщения на НИП на поверхности Земли можно воспользоваться командой `send_data`:

```
sputnik.transmitter.send_data(MESSAGE_SMS, message)
```

Первый и второй параметр функции в этой миссии можно оставить пустыми, они обозначают получателя и источник сообщения. В сообщение нужно передать тот текст, который был выдан команде в начальных условиях к миссии.

Радиоканал работает как очередь с подтверждением — как только вы передадите сообщение в подсистему радиосвязи, она будет пытаться переслать сообщение до тех пор, пока оно не будет полностью принято на НИП. Все последующие сообщения будут

добавляться в очередь.

Сообщение может быть доставлено на НИП только когда канал передачи становится больше 0, это возможно, когда НИП попадает в угол раскрыва антенны (биссектрисса угла совпадает с направлением ориентации аппарата). Поэтому в данной миссии вам нужно будет не только погасить начальное вращение аппарата, но и сориентировать его правильным образом.

Для решения данной миссии мы рекомендуем вам обратиться к разделу 2.3 «Энергетический расчет», разделу 2.4 «Тепловой расчет» и разделу 2.5 «Расчет информационного обмена» большого руководства по конструированию (http://orbitagame.ru/nti-2/media/pdf/orbita2.pdf).

РЕШЕНИЕ:

Для решения данной задачи нужно рассчитать константы для заданных параметров (по аналогии с предыдущей задачей) и написать следующую программу полета:

```
t = 508.40158921924325
w = -0.0621524626414
M0 = -4.3302006724e-05
M = 0.000001
dw = 0.00001

mode = 'rotate'
sputnik.orientation.set_motor_moment(AXIS_Z, M0);
sputnik.orientation.start_motor(AXIS_Z);
moment = True

while sputnik.cpu.run():

    if mode == 'rotate' and sputnik.cpu.get_flight_time() >= t:
        mode = 'ok'
        sputnik.orientation.stop_motor(AXIS_Z)
        moment = False
        sputnik.transmitter.set_state(STATE_ON)
        sputnik.transmitter.send_data(MESSAGE_SMS,
"b4OYKYLjBpzwmyan7BGATOd")

    if mode == 'ok':
        av = sputnik.orientation.get_angular_velocity(AXIS_Z)
        if abs(av - w) < dw:
            if moment:
                sputnik.orientation.stop_motor(AXIS_Z)
                moment = False
        elif not moment:
            sputnik.orientation.start_motor(AXIS_Z)
            moment = True
            if av > w:
                sputnik.orientation.set_motor_moment(AXIS_Z, -M)
            else:
                sputnik.orientation.set_motor_moment(AXIS_Z, M)
```

Задача 2.6 «Орбитальный маневр» (макс. 2300 баллов)

Следующая тренировочная миссия «Орбитальный маневр» дает возможность отработать смену аппаратом орбиты. Вам будет необходимо рассчитать массу топлива и написать программу полета, которая позволит аппарату сменить орбиту.

2.6.1. Условия победы

За успешное решение этой миссии вы получаете победные баллы. В миссии возможны следующие достижения:

Первые на орбите — Выполнить миссию первыми (300 балл.)

Вторые на орбите — Выполнить миссию вторыми (280 балл.)

Третьи на орбите — Выполнить миссию третьими (250 балл.)

Идеальное исполнение — Выполнить миссию с первой попытки (1000 балл.)

Высокая надежность — Выполнить миссию со второй-пятой попытки (400 балл.)

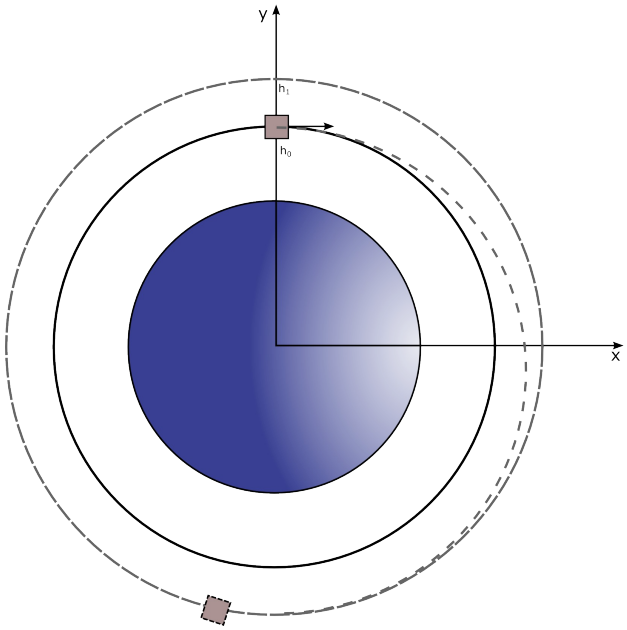
Миссия выполнена — Выполнить миссию с третьей и более попытки (100 балл.)

Точный расчет — Переход на новую орбиту осуществлен с ошибкой менее 1 км (1000 балл.)

2.6.2. Постановка задачи

Для решения многих реальных задач бывает необходимо перевести КА с одной постоянной орбиты на другую.

КА движется по круговой орбите с заданной высотой h_0 в плоскости X0Y. В этой миссии аппарат не имеет начальной скорости вращения. Необходимо запрограммировать аппарат так, чтобы он перешел на другую круговую орбиту h_1. Аппарат должен совершить **один виток по новой орбите** с отклонением не более **5 км**.



Анализ телеметрии после неудачного запуска позволит исправить ошибки, допущенные при расчете или программе полета КА.

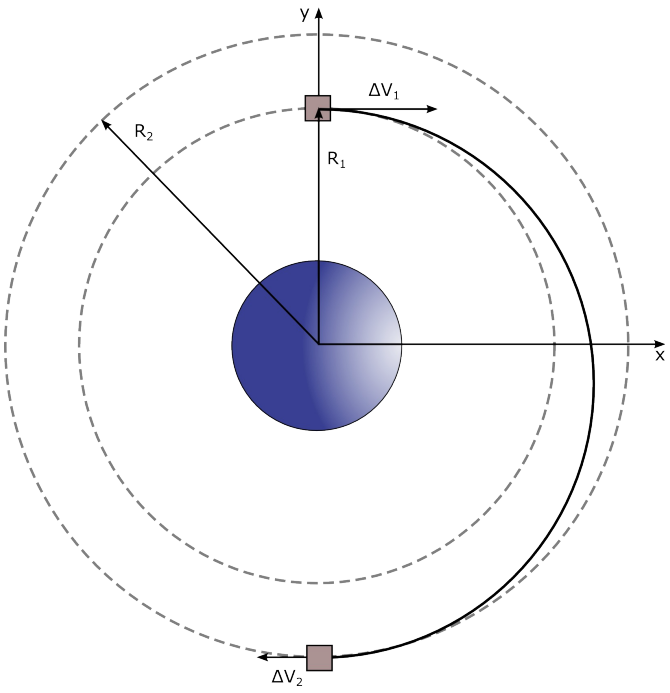
В данной миссии вам не потребуется конструировать аппарат. КА оснащен двигателем и небольшим топливным баком (объем топлива до 1 л). Вам нужно будет рассчитать и задать требуемую массу топлива, а также совершить двухимпульсный переход между двумя орбитами.

2.6.3. Исходные данные

Параметр	Пояснение	Значение
G	Гравитационная постоянная	$6{,}6742 \cdot 10^{-11}$ Н м^2/кг^2
M	Масса Земли	$5{,}9726 \cdot 10^{24}$ кг
R	Радиус Земли	6 371 032 м
h_0	Высота стартовой орбиты	См. уникальные условия миссии, м
h_1	Высота итоговой орбиты	См. уникальные условия миссии, м
m	Масса КА (без топлива)	6,4 кг
V_f	Максимальный объем топлива	1 л
r	Плотность топлива	1185 кг/м^3
$\mathbf{v_{обр}}$	Стартовая орбитальная скорость	Вычисляется по формуле: $v_{орб} = \sqrt{\frac{GM}{R + h_{орб}}}$, м/с
$I_{уд}$	Удельный импульс двигательной установки	2750 м/с
$\dot{m}$	Максимальный массовый расход топлива	0,009 кг/с
ω_0	Начальная угловая скорость	**0** °/с
$M_{макс}$	Предельный момент маховика СУОС	0,000023 Н м
I_z	Момент инерции кубического КА (в предположении, что масса распределена равномерно по его объему)	Вычисляется по формуле: $I_Z = \frac{1}{12}(2a^2)m$, кг м^2
a	Сторона грани кубического аппарата	0,1895 м
ε	Угловое ускорение КА	Вычисляется по формуле: $\varepsilon = \frac{M}{I_Z}$, °/с^2 где M — момент, с которым работает маховик.

2.6.4. Изменение орбиты КА

Для перехода КА на другую орбиту используется подсистема изменения орбиты, которая состоит из двигательной установки и топливных баков. Двигательная установка может выдавать реактивные импульсы. Сила тяги двигателя вычисляется по формуле:



$F_T = \dot{m} I_{уд}$, где $\dot{m}$ – массовый расход компонент топлива, кг/с, а $I_{уд}$ – удельный импульс двигательной установки, м/с. Параметры двигателя известны, их нужно учитывать при конструировании аппарата. Система изменения орбиты имеет топливные баки заданного объема. При конструировании аппарата нужно указать объем топлива, который необходимо залить в баки. При этом нужно помнить, что плотность топлива равна 1185 кг/м^3.

Для изменения орбиты КА можно использовать стандартный двухимпульсный переход. Суть этого перехода состоит в выдаче двух кратковременных импульсов — при сходе с орбиты и при выходе на новую орбиты. Двигатель включается на короткое время и изменяет скорость движения космического аппарата. Обратите внимание, что между импульсами аппарат должен поменять ориентацию на 180°, чтобы вторй импульс двигателя был направлен в противоположную сторону.

Можно предположить, что время, в течение которого двигательная установка выдает реактивный импульс значительно меньше, чем время движения аппарата между орбитами. В этом случае можно рассчитать параметры этих двух импульсов — при сходе с орбиты и при выходе на новую орбиту.

Каждый импульс характеризуется кратковременным увеличением скорости:

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

$$\Delta V_2 = \sqrt{\frac{G M}{R_2}} \left(1 - \sqrt{\frac{2 R_1}{R_1 + R_2}} \right),$$

где R_1 и R_2 — радиусы стартовой и итоговой орбит. Суммарное изменение скорости КА равно: $\Delta V_\Sigma = \Delta V_1 + \Delta V_2$.

Используя формулу Циолковского можно определить, какая масса топлива требуется для совершения этих импульсов:

$$m_{топл} = m \left(1 - e^{-\frac{\Delta V_\Sigma}{I_{уд}}} \right),$$

где m_0 — полная масса аппарата вместе с топливом.

При конструировании аппарата и выборе массы топлива очень важно провести эти расчеты предельно точно, даже несколько лишних килограмм топлива повлияют на точность выхода но новую орбиту.

При расчете орбит мы рекомендуем использовать Баллистический калькулятор.

2.6.5. Управление двигателем КА

Управление двигателем осуществляется через объект `sputnik.engine` в программе полета КА. Если подсистема изменения орбиты включена (состояние `STATE_ON`), можно выполнять следующие операции:

- `set_traction(t)` — задавать уровень массового расхода топлива t в кг/с;
- `start_engine()` — запустить тягу;
- `stop_engine()` — остановить тягу.

РЕШЕНИЕ:

Для решения этой задачи необходимо аккуратно произвести расчеты для заданных начальных условий и написать приблизительно такую программу полета:

```
 1. import math
 2.
 3. fuel_mass = 0.5
 4. total_mass = 6.4 + fuel_mass
 5. v = 6.8
 6. u = 2790.0
 7. G = 6.6742e-11
 8. R_z = 6371032.0
 9. M_z = 5.9726e24
10. target_height = 700.0 * 1000.0
11. R1 = R_z + 600.0 * 1000.0
12. R2 = R_z + target_height
13. M1 = 0.00002
14. M2 = 0.00001
15. Traction = 0.009
16. dw = 0.00001
17.
18. dv1 = math.sqrt(G*M_z / R1) * (math.sqrt(2 * R2 / (R1 + R2)) - 1)
19. dv2 = math.sqrt(G*M_z / R2) * (1 - math.sqrt(2 * R1 / (R1 + R2)))
20. dv = dv1 + dv2
21. need_fuel = total_mass * (1 - math.exp(-dv / u))
22.
23. stage = 1
24. start_velocity = sputnik.navigation.get_transversal_velocity()
25. sputnik.engine.set_traction(Traction)
26. sputnik.engine.start_engine()
27.
28. while sputnik.cpu.run():
29.     if stage == 1:
```

```
30.          v = sputnik.navigation.get_transversal_velocity()
31.          if abs(v - start_velocity) > dv1:
32.              sputnik.engine.stop_engine()
33.              stage = 2
34.      elif stage == 2:
35.          sputnik.orientation.set_motor_moment(AXIS_Z, M1)
36.          sputnik.orientation.start_motor(AXIS_Z)
37.          stage = 3
38.      elif stage == 3:
39.          oa = sputnik.orientation.get_angle(AXIS_Z)
40.          if oa >= 90:
41.              sputnik.orientation.set_motor_moment(AXIS_Z, -M1)
42.              stage = 4
43.      elif stage == 4:
44.          oa = sputnik.orientation.get_angle(AXIS_Z)
45.          if oa >= 180:
46.              sputnik.orientation.stop_motor(AXIS_Z)
47.              stage = 5
48.              moment = False
49.      else:
50.          av = sputnik.orientation.get_angular_velocity(AXIS_Z)
51.          if abs(av) < dw:
52.              if moment:
53.                  sputnik.orientation.stop_motor(AXIS_Z)
54.                  moment = False
55.          elif not moment:
56.              sputnik.orientation.start_motor(AXIS_Z)
57.              moment = True
58.              if av > 0:
59.                  sputnik.orientation.set_motor_moment(AXIS_Z, -M2)
60.              else:

61.                  sputnik.orientation.set_motor_moment(AXIS_Z, M2)
62.
63.          if stage == 5:
64.              h = sputnik.navigation.get_orbit_height()
65.              if h >= target_height:
66.                  start_velocity =
sputnik.navigation.get_transversal_velocity()
67.                  sputnik.engine.start_engine()
68.                  stage = 6
69.          elif stage == 6:
70.              v = sputnik.navigation.get_transversal_velocity()
71.              if abs(v - start_velocity) > dv2:
72.                  sputnik.engine.stop_engine()
73.                  stage = 7
```

Задача 2.7 «Съемка Земли из космоса» (макс. 12500 баллов)

Задача «Съемка Земли из космоса» посвящена съемке поверхности Земли из космоса малым космическим аппаратом. Вы должны будете сфотографировать объект на поверхности Земли и передать полученное изображение на наземный измерительный пункт (НИП), используя высокопроизводительную связь.

2.7.1. Условия победы

За успешное решение этой миссии вы получаете победные баллы. В миссии

возможны следующие достижения:

Первые на орбите — Выполнить миссию первыми (1500 балл.)

Вторые на орбите — Выполнить миссию вторыми (1200 балл.)

Третьи на орбите — Выполнить миссию третьими (1000 балл.)

Идеальное исполнение — Выполнить миссию с первой попытки (1000 балл.)

Высокая надежность — Выполнить миссию со второй-пятой попытки (400 балл.)

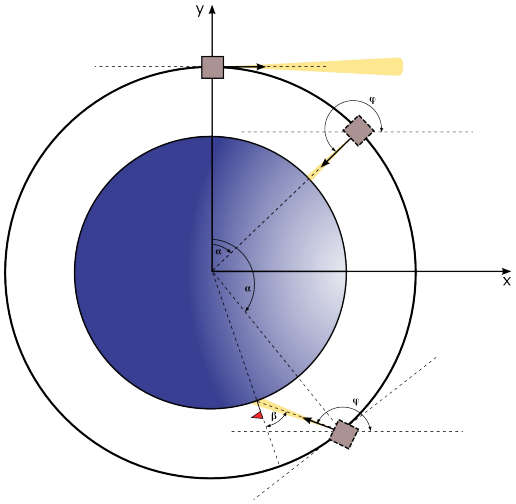
Миссия выполнена — Выполнить миссию с третьей и более попытки (100 балл.)

Высокое разрешение снимка — Разрешение снимка менее или равно 10 м на пиксель (5000 балл.)

Точное попадание — Угол отклонения от цели θ не превышает 0.01° (3000 балл.)

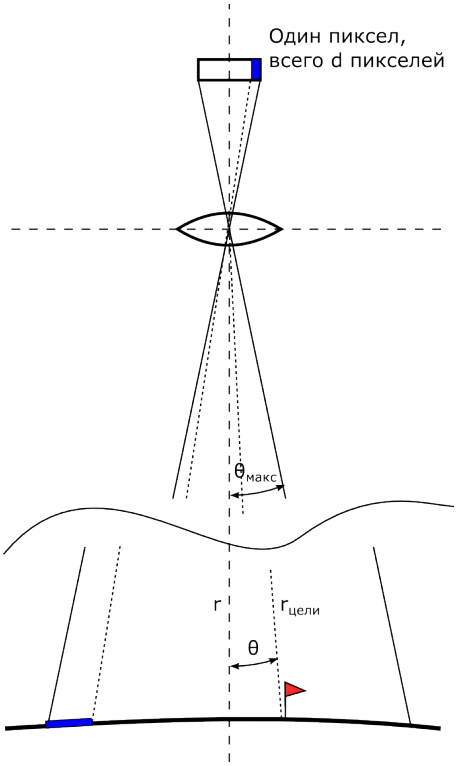
Вертикальная съемка — Угол отклонения от нормали β не превышает 0.01° (2000 балл.)

В вашем распоряжении будет 10 стартовых запусков. За каждый запуск после 10-го будет добавляться **штраф в 150 баллов за запуск**, и так до 20 дополнительных запусков (до -3000 баллов).



2.7.2. Исходные данные

Каждое конструкторское бюро получает уникальный вариант, который содержит: стартовую высоту орбиты и положения объекта на поверхности Земли и НИП, которые задается углами (0-359°), по аналогии с положением КА.



За время, которое отводится на миссию (6 часов) необходимо будет получить и передать на Землю наиболее **качественный** снимок. Качество снимка (и, соответственно, число победных баллов) зависит от трех параметров: разрешение снимка (в метрах на пиксель), угол отклонения цели от оси съемки (наведены ли мы точно на цель) и нормальность ориентации аппарата β в момент снимка (снимаем ли мы цель точно сверху). Чем выше разрешение и ближе ориентация аппарата к нормальному, тем выше число получаемых за миссию баллов. Если же объект вообще не попал в кадр, такой снимок не засчитывается.

В этой миссии вам придется полностью рассчитать и сконструировать спутник. Для этого мы

рекомендуем вам целиком изучить большое описание модели «Орбита: Частная космонавтика». Вам нужно будет выбрать полезную нагрузку КА — оптическую камеру. Параметры съемки описаны далее.

2.7.3. Съемка Земли из космоса

Съемка объектов на Земной поверхности производится с определенным разрешением. Для того, чтобы вычислить разрешение, необходимо рассмотреть, как устроена любая фотографическая съемка:

Камера имеет два значимых параметра: $\theta_{макс}$ — угол поля зрения камеры, т.е максимальный угол, объекты внутри которого попадают в кадр; d — число пикселей на матрице. Матрица рассматривается как одномерный набор пикселей. Каждый пиксел переходит в определенный отрезок на Земле, вычислить его можно следующим образом.

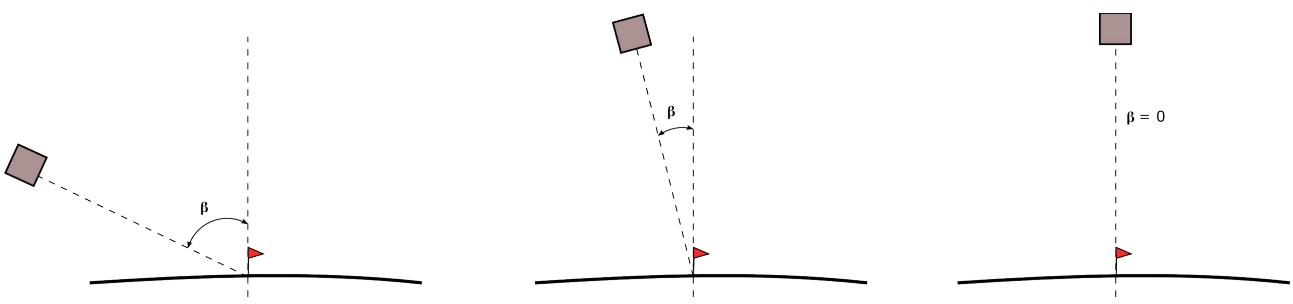
Будем считать, что КА находится на высоте r над поверхностью Земли во время съемки (высота орбиты). Тогда разрешение снимка вычисляется по формуле:

$$D = 2\frac{r \cdot \tan\left(\frac{\theta_{макс}}{2}\right)}{d}$$

Чем меньше метров на один пиксел, тем качественнее снимок. Таким образом, можно пытаться: 1) ставить камеру с наибольшим числом пикселей на матрице; 2) выбирать камеру наименьшим углом поля зрения (но будет сложнее поймать объект в кадре), а также 3) можно пробовать уменьшать расстояние до цели, например, переводя аппарат на более низкие орбиты.

Угол отклонения цели θ также очень важен при определении качества снимка — чем меньше этот угол, тем лучше. Чем меньше угол, тем выше качество снимка. Как минимум, этот угол не должен выходить за угол поля зрения камеры. Из всех данных, получаемых камерой, пока она включена, берутся те, что имеют наилучший угол отклонения.

Независимо от угла отклонения, важно учитывать, является ли ориентация аппарата нормальной по отношению к Земле.



В цель можно попасть очень точно (угол отклонения θ = 0°), но качество снимка

получится тем выше, чем более нормальная ориентация аппарата (в идеале $\beta = 0°$).

2.7.4. Управление камерой

Управление камерой производится через вызов специальных методов подсистемы полезной нагрузки. Для получения снимков камера должна быть включена (находиться в режиме `STATE_ON`). Для получения снимка можно использовать следующие методы:

- `camera.take_photo()` — камера делает моментальный снимок (продолжительность 0,05 с), метод возвращает номер блока памяти, в которой хранится снимок;
- `camera.start_shooting()` — камера начинает непрерывную съемку;
- `camera.stop_shooting()` — закончить непрерывную съемку, метод возвращает номер блока памяти, в которой хранится заснятая полоса поверхности;

Таким образом, у вас есть два варианта сделать снимок — в точке или получить отрезок. В любом случае на выходе должен получиться блок памяти, в котором хранится требуемый по условиям задания снимок. Его нужно передать на Землю через специальный вызов подсистемы высокопроизводительной связи:

```
slot = sputnik.take_photo()
...
sputnik.transmitter.send_photo(slot)
```

Как только верные данные будут переданы на НИП, миссия считается выполненной.

2.7.5. Объем памяти

В процессе съемки камера генерирует одиночный снимок или поток данных. Так или иначе, камера работает определенное время. Зная генерируемый объем данных в секунду, который указан в параметрах камеры, можно получить объем занимаемого блока памяти.

Важно рассчитать продолжительность съемки так, чтобы этот поток поместился в оперативную память подсистемы полезной нагрузки. Иначе из памяти будут выбрасываться все новые данные, и нужный объект может не попасть на снимок.

2.7.6. Изменение орбиты КА

Одним из способов сокращения расстояния до цели на поверхности является переход КА на более низкую орбиту, для чего используется подсистема изменения орбиты, которая состоит из двигательной установки и топливных баков.

Подробнее про способ изменения орбиты рассказывается в задании «Орбитальный

маневр».

При изменении орбиты необходимо учитывать атмосферу Земли, в которой на аппарат действует сила аэродинамического сопротивления (см. раздел 2.1.3 в большом руководстве к модели). Свойства атмосферы Земли можно найти в ГОСТ 4401-81 «Атмосфера станартная. Параметры» (таблицу легко найти в Википедии).

РЕШЕНИЕ:

Решение этой задачи потребует не только расчетов, аналогичных первым двум тренировочным миссиям, и написания программы полеты, но и конструирование аппарата из доступных подсистем. Для решения задачи достаточно использовать следующий аппарат:

- **Корпус:** Корпус для CubeSat-6U
- **Бортовая вычислительная система:** БЦВМ-1
- **Система электропитания:** СЭП с малым аккумулятором
- **Система навигации:** Навигатор-1
- **Система управления ориентацией и стабилизацией:** Система ориентации со средним управляющим моментом
- **Система обеспечения теплового режима:** СОТР с малым нагревателем
- **Система телеметрии:** Телеметрия с направленной антенной
- **Система высокопроизводительной связи:** Система связи X-диапазона узкой направленности
- **Система изменения орбиты:** нет
- **Полезная нагрузка:** Малая камера

Программа полета:

```
t = 508.401589219
w = -0.0621524626414
M0 = -9.7483082844e-05
M = 0.000001
dw = 0.00001
target_angle = 120.0

mode = 1
sputnik.orientation.set_motor_moment(AXIS_Z, M0);
sputnik.orientation.start_motor(AXIS_Z);
moment = True

while sputnik.cpu.run():

    if mode == 1 and sputnik.cpu.get_flight_time() >= t:
        mode = 2
        sputnik.orientation.stop_motor(AXIS_Z)
        moment = False
    elif mode == 2:
        a = sputnik.navigation.get_z_axis_angle()
        if abs(a - target_angle) <= 1:
            sputnik.camera.set_state(STATE_ON)
            sputnik.camera.start_shooting()
            mode = 3
```

```
    elif mode == 3:
        a = sputnik.navigation.get_z_axis_angle()
        if abs(a - target_angle) >= 1:
            slot = sputnik.camera.stop_shooting()
            sputnik.camera.set_state(STATE_OFF)
            sputnik.transmitter.set_state(STATE_ON)
            sputnik.transmitter.send_photo(slot)
            mode = 4
    elif mode == 4:
        if sputnik.cpu.mission_completed():
            break

    if mode >= 2:
        av = sputnik.orientation.get_angular_velocity(AXIS_Z)
        if abs(av - w) < dw:
            if moment:
                sputnik.orientation.stop_motor(AXIS_Z)
                moment = False
        elif not moment:
            sputnik.orientation.start_motor(AXIS_Z)
            moment = True
            if av > w:
                sputnik.orientation.set_motor_moment(AXIS_Z, -M)
            else:
                sputnik.orientation.set_motor_moment(AXIS_Z, M)
```

Задача 2.8. «SMS везде» (макс. 9200 баллов)

Задача «SMS везде» воссоздает работу спутника связи, который должен обеспечить прием и передачу сообщений между 18 наземными станциями (которые называются «0», «1», «2» и т.д.).

2.8.1. Условия победы

За успешное решение этой миссии вы получаете победные баллы. В миссии возможны следующие достижения:

Первые на орбите — Выполнить миссию первыми (1200 балл.)

Вторые на орбите — Выполнить миссию вторыми (1000 балл.)

Третьи на орбите — Выполнить миссию третьими (800 балл.)

Идеальный расчет — Выполнить миссию с первой попытки (3000 балл.)

Точный расчет — Выполнить миссию со второй-пятой попытки (200 балл.)

Миссия выполнена — Выполнить миссию с третьей и более попытки (1000 балл.)

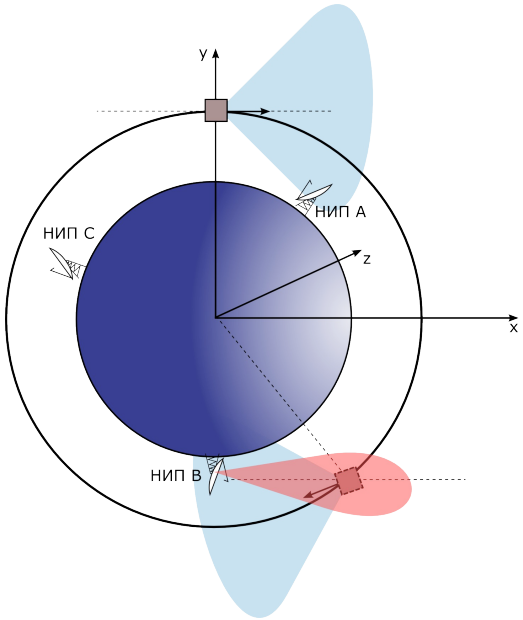
Высокая надежность — Переданы все 5 сообщений (5000 балл.)

Допустимая надежность — Переданы 3 и более сообщений (2000 балл.)

В вашем распоряжении будет 10 стартовых запусков. За каждый запуск после 10-го будет добавляться **штраф в 150 баллов за запуск**, и так до 20 дополнительных запусков (до -3000 баллов).

2.8.2. Постановка задачи

Каждое конструкторское бюро получает уникальный вариант, который содержит: стартовую высоту орбиты; список и названия наземных измерительных пунктов; таблицу сообщений для передачи (всего 5 сообщений).



КА должен последовательно доставить максимальное число сообщений, указанных в таблице. Как только аппарат получил сообщение от НИП, начинается отсчет времени доставки, которое не должно превысить допустимое время передачи. На миссию дается 6 часов полета аппарата.

Таблица сообщений имеет следующий формат:

Источник сообщения (НИП)	Получатель сообщения (НИП)	Объем данных сообщения (КБ)	Допустимая задержка передачи, с
0	3	20	6212
3	2	26	6095
...			

В этой миссии вам придется полностью рассчитать и сконструировать спутник. Для этого мы рекомендуем вам целиком изучить большое описание модели «Орбита: Частная космонавтика» (http://orbitagame.ru/nti-2/media/pdf/orbita2.pdf).

2.8.3. Получение и отправка сообщений

Для приема сообщений КА нужно обращаться к подсистеме высокопроизводительной связи `transmitter`. Прежде всего необходимо включить подсистему, изменив ее режим на `STATE_ON`:

```
sputnik.transmitter.set_state(STATE_ON)
```

Для принятия сообщения из НИП на поверхности Земли необходимо воспользоваться командой `receive`. Эта функция принимает единственный параметр — номер НИП, с которого ожидается сообщение. Эта команда начинает прием сообщения. Для

проверки наличия принятого сообщения необходимо использовать функцию `get_progress`, которая возвращает, какой процент сообщения был принят (от 0 до 100). После того как сообщение получено, можно использовать функцию `get_message`, чтобы прочитать полученное сообщение.

```
sputnik.transmitter.receive(msg_from)
...
if sputnik.transmitter.get_progress(msg_from) == 100.0:
    msg = sputnik.transmitter.get_message(msg_from)
    msg_from = msg.sender
    msg_to = msg.receiver
    data = msg.data
    timeout = msg.timeout
    ...
```

Обратите внимание: между командой на прием сообщения (`receive`) и фактом приема сообщения обязательно должно пройти какое-то время, ведь сообщение должно накопиться в приемном буфере подсистемы высокопроизводительной связи.

Возможен только последовательный прием сообщений КА. При этом допускается одновременные прием и пересылка сообщения. Канал связи при этом сперва используется для приема информации КА, а только затем, если осталась незадействованная полоса связи, для отправки сообщений.

Для последующей отправки сообщения на НИП на поверхности Земли можно воспользоваться командой `send_data`:

```
sputnik.transmitter.send_message(MESSAGE_SMS, data, msg_to, msg_from, timeout)
```

В данной миссии принципиально важно указывать, от какого к какому НИП отправляется сообщение, иначе оно может не дойти до адресата. **Остерегаем вас** от использования в этой миссии сообщений с многими адресатами (когда msg_to равно -1). В этом случае множество НИП-ов получат не те сообщения, которые ожидают, и баллы за миссию можно вообще не получить.

РЕШЕНИЕ:

Решение этой задачи потребует не только расчетов, аналогичных первым двум тренировочным миссиям, и написания программы полеты, но и конструирование аппарата из доступных подсистем. Для решения задачи достаточно использовать следующий аппарат:

- **Корпус:** Корпус для CubeSat-3U
- **Бортовая вычислительная система:** БЦВМ-1
- **Система электропитания:** СЭП с малым аккумулятором
- **Система навигации:** Навигатор-1
- **Система управления ориентацией и стабилизацией:** Система ориентации со средним управляющим моментом

- **Система обеспечения теплового режима:** СОТР с малым нагревателем
- **Система телеметрии:** Телеметрия с ненаправленной антенной
- **Система высокопроизводительной связи:** Система УКВ-связи
- **Система изменения орбиты:** нет
- **Полезная нагрузка:** нет

Программа полета:

```
initial_height = 645000.0

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

max_orientation_torsion = 0.0026 / 4

heater_power = 4.0

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

radio_angle_precision = 10.0

routing = [(6, 10), (16, 3), (11, 14), (1, 5), (9, 17)]

station_angles = [6, 27, 52, 61, 88, 116, 127, 157, 164, 197, 213, 230,
241, 273, 291, 316, 322, 353]

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 =
putnik.orientation.get_angular_velocity(AXIS_Z)
            if abs(angular_velocity - self.desired_angular_velocity) >
angular_velocity_precision:
                if not self.active:
```

```
49.                     self.active = True
50.                     if angular_velocity > self.desired_angular_velocity:
51.                         torsion = -max_orientation_torsion
52.                     else:
53.                         torsion = max_orientation_torsion
54.                     sputnik.orientation.start_motor(AXIS_Z)
55.                     sputnik.orientation.set_motor_moment(AXIS_Z,
torsion)
56.             elif self.active:
57.                 sputnik.orientation.stop_motor(AXIS_Z)
58.                 self.active = False
59.
60. damping_mode = DampingMode()
61.
62. class OrientationMode(object):
63.     def __init__(self):
64.         self.enabled = False
65.         self.active = False
66.         self.differential = True
67.         self.desired_angle = 0.0
68.         self.desired_angular_velocity = 0.0
69.         self.max_angular_acceleration = 0.0
70.         self.inertia_moment = 0.0
71.
72.     def run(self):
73.         if self.enabled:
74.             angle = sputnik.orientation.get_angle(AXIS_Z)
75.             angular_velocity =
sputnik.orientation.get_angular_velocity(AXIS_Z)
76.             desired_angle_change =
normalize_angle_difference(self.desired_angle - angle)
77.             if ((abs(angular_velocity - self.desired_angular_velocity) >
angular_velocity_precision) or
78.                 (abs(desired_angle_change) > orient_angle_precision)):
79.                 proportional_coefficient = 8 *
self.max_angular_acceleration / 180
80.                 differential_coefficient = 8 *
math.sqrt(self.max_angular_acceleration / 360)
81.                 angular_acceleration = desired_angle_change *
proportional_coefficient
82.                 if self.differential:
83.                     angular_acceleration -= (angular_velocity -
self.desired_angular_velocity) * differential_coefficient
84.                 torsion = self.inertia_moment * angular_acceleration
85.                 if torsion > max_orientation_torsion:
86.                     torsion = max_orientation_torsion
87.                 if torsion < -max_orientation_torsion:
88.                     torsion = -max_orientation_torsion
89.                 sputnik.orientation.set_motor_moment(AXIS_Z, torsion)
90.                 if not self.active:
91.                     self.active = True
92.                     sputnik.orientation.start_motor(AXIS_Z)
93.             elif self.active:
94.                 sputnik.orientation.stop_motor(AXIS_Z)
95.                 self.active = False
96.
97. orientation_mode = OrientationMode()
98.
99. class TargetingMode(object):
100.     def __init__(self):
101.         self.enabled = False
```

```
   102.         self.active = False
   103.         self.initial_nav_angular_velocity = math.degrees(math.sqrt((G *
earth_mass) / ((initial_height + earth_radius) ** 3)))
   104.
   105.     def run(self):
   106.         if self.enabled:
   107.             flight_time = sputnik.cpu.get_flight_time()
   108.             nav_angle = sputnik.navigation.get_z_axis_angle()
   109.             orientation_mode.desired_angle = normalize_angle(270 -
nav_angle)
   110.             orientation_mode.desired_angular_velocity =
-self.initial_nav_angular_velocity
   111.             if not self.active:
   112.                 self.active = True
   113.
   114. targeting_mode = TargetingMode()
   115.
   116. state = 0
   117.
   118. message_index = 0
   119.
   120. message = None
   121.
   122. heating = False
   123.
   124. while sputnik.cpu.run():
   125.     damping_mode.run()
   126.     orientation_mode.run()
   127.     targeting_mode.run()
   128.
   129.     temperature = sputnik.heat_control.get_temperature()
   130.     if (temperature < 285) and not heating:
   131.         heating = True
   132.         sputnik.heat_control.set_power(heater_power)
   133.         sputnik.heat_control.start_heating()
   134.     elif (temperature > 300) and heating:
   135.         heating = False
   136.         sputnik.heat_control.set_power(0)
   137.         sputnik.heat_control.stop_heating()
   138.
   139.     if state == 0:
   140.         sputnik.telemetry.set_state(STATE_ON)
   141.         sputnik.navigation.set_state(STATE_ON)
   142.         sputnik.orientation.set_state(STATE_ON)
   143.         sputnik.heat_control.set_state(STATE_ON)
   144.         initial_angular_velocity =
sputnik.orientation.get_angular_velocity(AXIS_Z)
   145.         flight_time = sputnik.cpu.get_flight_time()
   146.         stabilization_begin_time = flight_time
   147.         damping_mode.enabled = True
   148.         state = 1
   149.         continue
   150.
   151.     if (state == 1) and not damping_mode.active:
   152.         damping_mode.enabled = False
   153.         orientation_mode.max_angular_acceleration =
abs(initial_angular_velocity) / (sputnik.cpu.get_flight_time() -
stabilization_begin_time)
   154.         orientation_mode.inertia_moment = max_orientation_torsion /
orientation_mode.max_angular_acceleration
   155.         state = 10
```

```
   156.         continue
   157.
   158.     if state == 10:
   159.         targeting_mode.enabled = True
   160.         state = 11
   161.         continue
   162.
   163.     if (state == 11) and targeting_mode.active:
   164.         orientation_mode.enabled = True
   165.         state = 12
   166.         continue
   167.
   168.     if (state == 12) and not orientation_mode.active:
   169.         sputnik.transmitter.set_state(STATE_ON)
   170.         sputnik.transmitter.receive(routing[message_index][0])
   171.         state = 20
   172.         continue
   173.
   174.     if (state == 20) and
(sputnik.transmitter.get_progress(routing[message_index][0]) >= 100.0):
   175.         message = sputnik.transmitter.get_message(routing[message_index]
[0])
   176.         state = 30
   177.         continue
   178.
   179.     if (state == 30) and
(abs(normalize_angle_difference(station_angles[routing[message_index][1]] -
sputnik.navigation.get_z_axis_angle())) < radio_angle_precision):
   180.         sputnik.transmitter.send_data(MESSAGE_SMS, message.data,
routing[message_index][1], routing[message_index][0])
   181.         message_index += 1
   182.         if message_index == len(routing):
   183.             state = 40
   184.             continue
   185.         else:
   186.             sputnik.transmitter.receive(routing[message_index][0])
   187.             state = 20
   188.             continue
```

Задача 2.9 «Инспекция спутника» (макс. 28000 баллов)

Задача «Инспекция спутника» рассматривает ситуацию съемки объекта, который движется по другой орбите вокруг Земли. Необходимо сфотографировать объект в максимальном разрешении и передать полученное изображение на наземный измерительный пункт (НИП), используя высокопроизводительную связь.

2.9.1. Условия победы

За успешное решение этой миссии вы получаете победные баллы. В миссии возможны следующие достижения:

Первые на орбите — Выполнить миссию первыми (2000 балл.)

Вторые на орбите — Выполнить миссию вторыми (1800 балл.)

Третьи на орбите — Выполнить миссию третьими (1600 балл.)

С первой попытки — Выполнить миссию с первой попытки (6000 балл.)

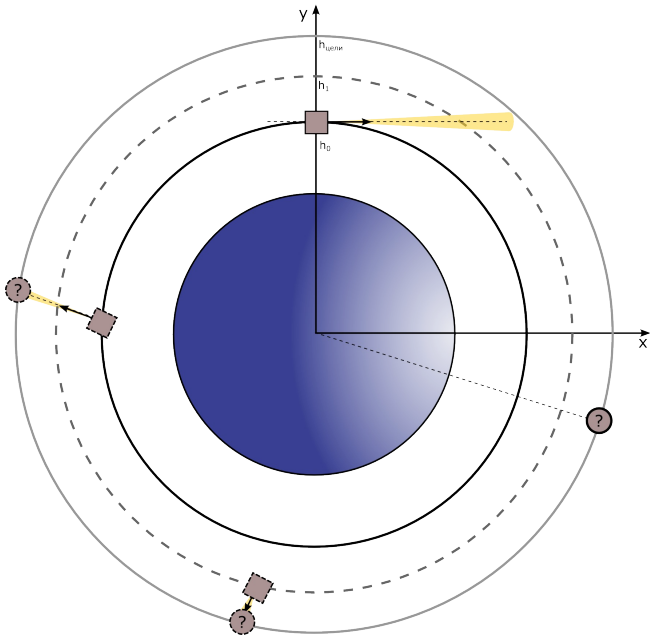
Высокая надежность — Выполнить миссию со второй-пятой попытки (4000 балл.)

Миссия выполнена — Выполнить миссию с шестой и более попытки (2000 балл.)

Высокое разрешение снимка — Разрешение снимка менее или равно 10 м на пиксель (10000 балл.)

Подкрался к цели — Расстояние до цели меньше или равно 1 км (10000 балл.)

В вашем распоряжении будет 10 стартовых запусков. За каждый запуск после 10-го будет добавляться **штраф в 150 баллов за запуск**, и так до 20 дополнительных запусков (до -3000 баллов).



2.9.2. Постановка задачи

Каждое конструкторское бюро получает уникальный вариант, который содержит: стартовую высоту орбиты и орбиту цели, а также положение НИП.

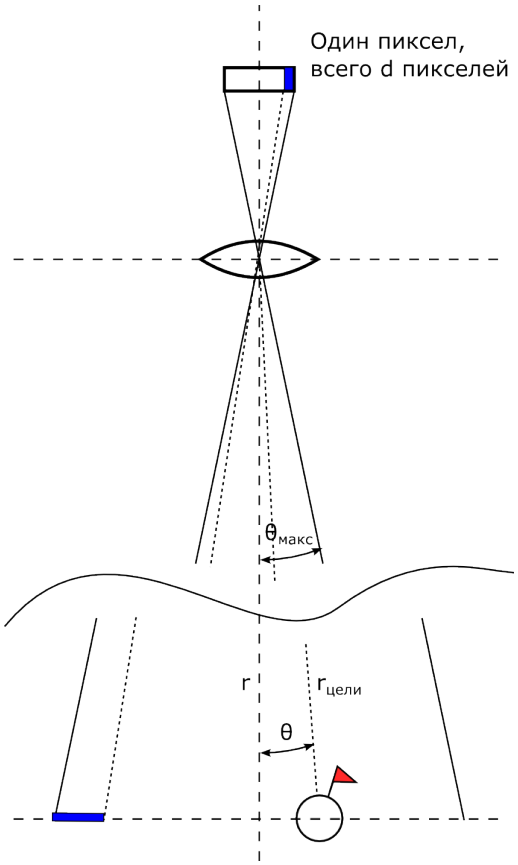
За время, которое отводится на миссию (6 часов) необходимо будет получить и передать на Землю наиболее **качественный** снимок инспектируемого объекта.

Качество снимка (число победных баллов) зависит от его разрешения (в метрах на пиксель) и угла отклонения цели от оси съемки (наведены ли мы точно на цель). Чем выше разрешение и выше точность попадания, тем выше число получаемых за миссию баллов. Если объект вообще не попал в кадр, такой снимок не засчитывается.

В этой миссии вам придется полностью рассчитать и сконструировать спутник. Для этого мы рекомендуем вам целиком изучить большое описание модели «Орбита: Частная космонавтика» (http://orbitagame.ru/nti-2/media/pdf/orbita2.pdf). Вам нужно будет выбрать полезную нагрузку КА — оптическую камеру.

2.9.3. Съемка объектов космосе

Съемка объектов производится с определенным разрешением. Для того, чтобы вычислить разрешение, необходимо рассмотреть, как устроена любая фотографическая съемка:



Камера имеет два значимых параметра: $\theta_{макс}$ — угол поля зрения камеры, т.е максимальный угол, объекты внутри которого попадают в кадр; d — число пикселей на матрице. Матрица рассматривается как одномерный набор пикселей. Каждый пиксел переходит в определенный отрезок на Земле, вычислить его можно следующим образом.

Будем считать, что КА находится на высоте r над поверхностью Земли во время съемки (высота орбиты). Тогда разрешение снимка вычисляется по формуле:

$$D = 2\frac{r_{цели}\cdot\cos\theta\cdot\tan\left(\frac{\theta_{макс}}{2}\right)}{d} = 2\frac{r\cdot\tan\left(\frac{\theta_{макс}}{2}\right)}{d}$$

Чем меньше метров на один пиксел, тем качественнее снимок. Таким образом, можно пытаться: 1) ставить камеру с наибольшим числом пикселей на матрице; 2) выбирать камеру наименьшим углом поля зрения (но будет сложнее поймать объект в кадре), а также 3) уменьшать расстояние до цели.

Угол отклонения цели θ также очень важен при определении качества снимка — чем меньше этот угол, тем лучше. Чем меньше угол, тем выше качество снимка. Как минимум, этот угол не должен выходить за угол поля зрения камеры. Из всех данных, получаемых камерой, пока она включена, берутся те, что имеют наилучший угол отклонения.

2.9.4. Изменение орбиты КА

Для перехода КА на другую орбиту используется подсистема изменения орбиты, которая состоит из двигательной установки и топливных баков.

Подробнее про способ изменения орбиты рассказывается в задании «Орбитальный маневр».

2.9.5. Управление камерой

Управление камерой производится через вызов специальных методов подсистемы полезной нагрузки.

Подробнее про способ изменения орбиты рассказывается в задании «Съемка Земли

из космоса».

РЕШЕНИЕ:

Решение этой задачи потребует не только расчетов, аналогичных первым двум тренировочным миссиям, и написания программы полеты, но и конструирование аппарата из доступных подсистем. Для решения задачи достаточно использовать следующий аппарат:

- **Корпус:** Корпус MicroSat-Case
- **Бортовая вычислительная система:** БЦВМ-2
- **Система электропитания:** СЭП с большим аккумулятором
- **Система навигации:** Навигатор-1
- **Система управления ориентацией и стабилизацией:** Система ориентации с большим управляющим моментом
- **Система обеспечения теплового режима:** СОТР со средним нагревателем
- **Система телеметрии:** Телеметрия с ненаправленной антенной
- **Система высокопроизводительной связи:** Система связи X-диапазона узкой направленности
- **Система изменения орбиты:** Двигатель с большой тягой и средним баком
- **Полезная нагрузка:** нет

Программа полета:

```
initial_inspector_height = 182000.0
target_height = 790000.0

initial_target_angle = 61.0
ground_station_angle = 121.0

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

max_orientation_torsion = 0.0165

max_engine_traction = 0.037
engine_specific_impulse = 2705.0

dry_mass = 1.7+10.0+6.6 + 0.6 + 0.2 + 0.4 + 3.0 + 8 + 1.5 + 1.5

first_burn_tune = 0.00
second_burn_tune = 0.00

load_enable_interval = 1.5
radio_enable_interval = 200

heater_power = 5.0

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
```

```
    33.     while normalized_angle < 0:
    34.         normalized_angle += 360
    35.     while normalized_angle >= 360:
    36.         normalized_angle -= 360
    37.     return normalized_angle
    38.
    39. def normalize_angle_difference(angle_difference):
    40.     normalized_angle_difference = angle_difference
    41.     while normalized_angle_difference < -180:
    42.         normalized_angle_difference += 360
    43.     while normalized_angle_difference >= 180:
    44.         normalized_angle_difference -= 360
    45.     return normalized_angle_difference
    46.
    47. class DampingMode(object):
    48.     def __init__(self):
    49.         self.enabled = False
    50.         self.active = False
    51.         self.desired_angular_velocity = 0.0
    52.
    53.     def run(self):
    54.         if self.enabled:
    55.             angular_velocity =
sputnik.orientation.get_angular_velocity(AXIS_Z)
    56.             if abs(angular_velocity - self.desired_angular_velocity) >
angular_velocity_precision:
    57.                 if not self.active:
    58.                     self.active = True
    59.                     if angular_velocity > self.desired_angular_velocity:
    60.                         torsion = -max_orientation_torsion
    61.                     else:
    62.                         torsion = max_orientation_torsion
    63.                     sputnik.orientation.start_motor(AXIS_Z)
    64.                     sputnik.orientation.set_motor_moment(AXIS_Z,
torsion)
    65.             elif self.active:
    66.                 sputnik.orientation.stop_motor(AXIS_Z)
    67.                 self.active = False
    68.
    69. damping_mode = DampingMode()
    70.
    71. class OrientationMode(object):
    72.     def __init__(self):
    73.         self.enabled = False
    74.         self.active = False
    75.         self.differential = True
    76.         self.desired_angle = 0.0
    77.         self.desired_angular_velocity = 0.0
    78.         self.max_angular_acceleration = 0.0
    79.         self.inertia_moment = 0.0
    80.
    81.     def run(self):
    82.         if self.enabled:
    83.             angle = sputnik.orientation.get_angle(AXIS_Z)
    84.             angular_velocity =
sputnik.orientation.get_angular_velocity(AXIS_Z)
    85.             desired_angle_change =
normalize_angle_difference(self.desired_angle - angle)
    86.             if ((abs(angular_velocity - self.desired_angular_velocity) >
angular_velocity_precision) or
    87.                 (abs(desired_angle_change) > orient_angle_precision)):
```

```
    88.                 proportional_coefficient = 8 *
self.max_angular_acceleration / 180
    89.                 differential_coefficient = 8 *
math.sqrt(self.max_angular_acceleration / 360)
    90.                 angular_acceleration = desired_angle_change *
proportional_coefficient
    91.                 if self.differential:
    92.                     angular_acceleration -= (angular_velocity -
self.desired_angular_velocity) * differential_coefficient
    93.                 torsion = self.inertia_moment * angular_acceleration
    94.                 if torsion > max_orientation_torsion:
    95.                     torsion = max_orientation_torsion
    96.                 if torsion < -max_orientation_torsion:
    97.                     torsion = -max_orientation_torsion
    98.                 sputnik.orientation.set_motor_moment(AXIS_Z, torsion)
    99.                 if not self.active:
   100.                     self.active = True
   101.                     sputnik.orientation.start_motor(AXIS_Z)
   102.             elif self.active:
   103.                 sputnik.orientation.stop_motor(AXIS_Z)
   104.                 self.active = False
   105.
   106. orientation_mode = OrientationMode()
   107.
   108. class PhasingMode(object):
   109.     def __init__(self):
   110.         self.enabled = False
   111.         self.active = False
   112.         self.angle_alignment = 0.0
   113.         self.target_angle = 0.0
   114.         self.initial_nav_angular_velocity = math.degrees(math.sqrt((G *
earth_mass) / ((initial_inspector_height + earth_radius) ** 3)))
   115.         self.target_angular_velocity = math.degrees(math.sqrt((G *
earth_mass) / ((target_height + earth_radius) ** 3)))
   116.
   117.     def run(self):
   118.         if self.enabled:
   119.             flight_time = sputnik.cpu.get_flight_time()
   120.             self.target_angle = normalize_angle(initial_target_angle +
flight_time * self.target_angular_velocity)
   121.             nav_angle = sputnik.navigation.get_z_axis_angle()
   122.             orientation_mode.desired_angle = normalize_angle(360 -
nav_angle)
   123.             orientation_mode.desired_angular_velocity =
-self.initial_nav_angular_velocity
   124.             self.angle_alignment =
normalize_angle_difference(self.target_angle - nav_angle)
   125.             if not self.active:
   126.                 self.active = True
   127.
   128. phasing_mode = PhasingMode()
   129.
   130. class TrackingMode(object):
   131.     def __init__(self):
   132.         self.enabled = False
   133.         self.active = False
   134.         self.target_angle = 0.0
   135.         self.target_direction = 0.0
   136.         self.sample_time = 0.0
   137.         self.target_direction_angular_velocity = 0.0
```

```
   138.         self.target_angular_velocity = math.degrees(math.sqrt((G *
earth_mass) / ((target_height + earth_radius) ** 3)))
   139.
   140.     def run(self):
   141.         if self.enabled:
   142.             flight_time = sputnik.cpu.get_flight_time()
   143.             self.target_angle = normalize_angle(initial_target_angle +
flight_time * self.target_angular_velocity)
   144.             target_x = (target_height + earth_radius) *
math.sin(math.radians(self.target_angle))
   145.             target_y = (target_height + earth_radius) *
math.cos(math.radians(self.target_angle))
   146.             our_x = sputnik.navigation.get_x_coord()
   147.             our_y = sputnik.navigation.get_y_coord()
   148.             prev_target_direction = self.target_direction
   149.             prev_sample_time = self.sample_time
   150.             self.target_direction = math.degrees(math.atan2(target_y -
our_y, target_x - our_x))
   151.             self.sample_time = flight_time
   152.             if self.sample_time > prev_sample_time:
   153.                 self.target_direction_angular_velocity =
(self.target_direction - prev_target_direction) / (self.sample_time -
prev_sample_time)
   154.             orientation_mode.desired_angle = self.target_direction
   155.             orientation_mode.desired_angular_velocity =
self.target_direction_angular_velocity
   156.             if not self.active:
   157.                 self.active = True
   158.
   159. tracking_mode = TrackingMode()
   160.
   161. state = 0
   162.
   163. heating = False
   164.
   165. while sputnik.cpu.run():
   166.     damping_mode.run()
   167.     orientation_mode.run()
   168.     phasing_mode.run()
   169.     tracking_mode.run()
   170.
   171.     temperature = sputnik.heat_control.get_temperature()
   172.     if (temperature < 285) and not heating:
   173.         heating = True
   174.         sputnik.heat_control.set_power(heater_power)
   175.         sputnik.heat_control.start_heating()
   176.     elif (temperature > 300) and heating:
   177.         heating = False
   178.         sputnik.heat_control.set_power(0)
   179.         sputnik.heat_control.stop_heating()
   180.
   181.     if state == 0:
   182.         sputnik.telemetry.set_state(STATE_ON)
   183.         sputnik.navigation.set_state(STATE_ON)
   184.         sputnik.orientation.set_state(STATE_ON)
   185.         sputnik.heat_control.set_state(STATE_ON)
   186.         initial_angular_velocity =
sputnik.orientation.get_angular_velocity(AXIS_Z)
   187.         flight_time = sputnik.cpu.get_flight_time()
   188.         stabilization_begin_time = flight_time
   189.         damping_mode.enabled = True
```

```
   190.         state = 1
   191.         continue
   192.
   193.     if (state == 1) and not damping_mode.active:
   194.         damping_mode.enabled = False
   195.         orientation_mode.max_angular_acceleration =
abs(initial_angular_velocity) / (sputnik.cpu.get_flight_time() -
stabilization_begin_time)
   196.         orientation_mode.inertia_moment = max_orientation_torsion /
orientation_mode.max_angular_acceleration
   197.         state = 10
   198.         continue
   199.
   200.     if state == 10:
   201.         phasing_mode.enabled = True
   202.         state = 11
   203.         continue
   204.
   205.     if (state == 11) and phasing_mode.active:
   206.         orientation_mode.enabled = True
   207.         state = 12
   208.         continue
   209.
   210.     if (state == 12) and not orientation_mode.active:
   211.         transfer_angle_alignment = normalize_angle_difference(180 * (1 -
math.sqrt((1 + (initial_inspector_height + earth_radius) / (target_height +
earth_radius)) ** 3) / math.sqrt(8)))
   212.         transfer_delta_v = abs(math.sqrt(G * earth_mass /
(initial_inspector_height + earth_radius)) * (math.sqrt(2 * (target_height +
earth_radius) / ((target_height + earth_radius) + (initial_inspector_height +
earth_radius))) - 1))
   213.         sputnik.engine.set_state(STATE_ON)
   214.         transfer_start_fuel = sputnik.engine.get_fuel()
   215.         sputnik.engine.set_state(STATE_OFF)
   216.         transfer_end_fuel = ((dry_mass + transfer_start_fuel) /
math.exp(transfer_delta_v / engine_specific_impulse)) - dry_mass
   217.         if transfer_end_fuel < 0.0:
   218.             transfer_end_fuel = 0.0
   219.         transfer_burn_time = (transfer_start_fuel - transfer_end_fuel) /
(max_engine_traction / 2)
   220.         transfer_burn_angle = transfer_burn_time *
phasing_mode.initial_nav_angular_velocity
   221.         #transfer_delta_v = transfer_delta_v *
math.radians(transfer_burn_angle) / (2 *
math.sin(math.radians(transfer_burn_angle / 2))) + first_burn_tune
   222.         transfer_delta_v = transfer_delta_v + first_burn_tune
   223.         transfer_end_fuel = ((dry_mass + transfer_start_fuel) /
math.exp(transfer_delta_v / engine_specific_impulse)) - dry_mass
   224.         if transfer_end_fuel < 0.0:
   225.             transfer_end_fuel = 0.0
   226.         transfer_burn_time = (transfer_start_fuel - transfer_end_fuel) /
(max_engine_traction / 2)
   227.         transfer_burn_angle = transfer_burn_time *
phasing_mode.initial_nav_angular_velocity
   228.         state = 13
   229.         continue
   230.
   231.     if (state == 13) and
(abs(normalize_angle_difference(phasing_mode.angle_alignment -
transfer_angle_alignment - (transfer_burn_time / 2) *
(phasing_mode.initial_nav_angular_velocity -
```

```
phasing_mode.target_angular_velocity))) < navig_angle_precision):
    nav_angle = sputnik.navigation.get_z_axis_angle()
    transfer_center_angle = nav_angle + transfer_burn_angle / 2
    sputnik.engine.set_state(STATE_ON)
    sputnik.engine.set_traction(max_engine_traction / 2)
    sputnik.engine.start_engine()
    state = 14
    continue

if (state == 14) and (sputnik.engine.get_fuel() <=
transfer_end_fuel):
    sputnik.engine.set_traction(0)
    sputnik.engine.set_state(STATE_OFF)
    phasing_mode.enabled = False
    state = 20
    continue

if state == 20:
    orientation_mode.desired_angle = normalize_angle(360 -
normalize_angle(transfer_center_angle + 180))
    orientation_mode.desired_angular_velocity = 0
    state = 21
    continue

if (state == 21) and not orientation_mode.active:
    transfer_delta_v = abs(math.sqrt(G * earth_mass / (target_height
+ earth_radius)) * (1 - math.sqrt(2 * (initial_inspector_height +
earth_radius) / ((target_height + earth_radius) + (initial_inspector_height +
earth_radius)))))
    sputnik.engine.set_state(STATE_ON)
    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 / 2)
    transfer_burn_angle = transfer_burn_time *
(phasing_mode.target_angular_velocity - (transfer_delta_v / (target_height +
earth_radius)))
    #transfer_delta_v = transfer_delta_v *
math.radians(transfer_burn_angle) / (2 *
math.sin(math.radians(transfer_burn_angle / 2))) + second_burn_tune
    transfer_delta_v = transfer_delta_v + second_burn_tune
    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 / 2)
    transfer_burn_angle = transfer_burn_time *
(phasing_mode.target_angular_velocity - (transfer_delta_v / (target_height +
earth_radius)))
    state = 22
    continue

if (state == 22) and
(abs(normalize_angle_difference(sputnik.navigation.get_z_axis_angle() -
normalize_angle(transfer_center_angle + 180.0 - transfer_burn_angle / 2))) <
navig_angle_precision):
```

```
274.          sputnik.engine.set_state(STATE_ON)
275.          sputnik.engine.set_traction(max_engine_traction / 2)
276.          sputnik.engine.start_engine()
277.          state = 23
278.          continue
279.
280.      if (state == 23) and (sputnik.engine.get_fuel() <=
transfer_end_fuel):
281.          sputnik.engine.set_traction(0)
282.          sputnik.engine.set_state(STATE_OFF)
283.          phasing_mode.enabled = False
284.          state = 30
285.          continue
286.
287.      if state == 30:
288.          tracking_mode.enabled = True
289.          state = 31
290.          continue
291.
292.      if (state == 31) and tracking_mode.active:
293.          state = 32
294.          continue
295.
296.      if (state == 32) and not orientation_mode.active:
297.          state = 40
298.          continue
299.
300.      if (state == 40):
301.          sputnik.transmitter.set_state(STATE_ON)
302.          sputnik.camera.set_state(STATE_ON)
303.          load_enable_time = sputnik.cpu.get_flight_time()
304.          load_enable_angle = sputnik.navigation.get_z_axis_angle()
305.          sputnik.camera.start_shooting()
306.          state = 41
307.          continue
308.
309.      if (state == 41) and ((sputnik.cpu.get_flight_time() -
load_enable_time) >= load_enable_interval):
310.          slot = sputnik.camera.stop_shooting()
311.          sputnik.transmitter.send_photo(slot)
312.          sputnik.camera.set_state(STATE_OFF)
313.          state = 42
314.          continue
315.
316.      if (state == 42) and
(abs(normalize_angle_difference(load_enable_angle + 90 -
sputnik.navigation.get_z_axis_angle())) < navig_angle_precision):
317.          sputnik.transmitter.set_state(STATE_ON)
318.          sputnik.camera.set_state(STATE_ON)
319.          load_enable_time = sputnik.cpu.get_flight_time()
320.          load_enable_angle = sputnik.navigation.get_z_axis_angle()
321.          sputnik.camera.start_shooting()
322.          state = 43
323.          continue
324.
325.      if (state == 43) and ((sputnik.cpu.get_flight_time() -
load_enable_time) >= load_enable_interval):
326.          slot = sputnik.camera.stop_shooting()
327.          sputnik.transmitter.send_photo(slot)
328.          sputnik.camera.set_state(STATE_OFF)
329.          state = 44
```

```
330.         continue
331.
332.     if (state == 44) and
(abs(normalize_angle_difference(load_enable_angle + 90 -
sputnik.navigation.get_z_axis_angle())) < navig_angle_precision):
333.         sputnik.transmitter.set_state(STATE_ON)
334.         sputnik.camera.set_state(STATE_ON)
335.         load_enable_time = sputnik.cpu.get_flight_time()
336.         load_enable_angle = sputnik.navigation.get_z_axis_angle()
337.         sputnik.camera.start_shooting()
338.         state = 45
339.         continue
340.
341.     if (state == 45) and ((sputnik.cpu.get_flight_time() -
load_enable_time) >= load_enable_interval):
342.         slot = sputnik.camera.stop_shooting()
343.         sputnik.transmitter.send_photo(slot)
344.         sputnik.camera.set_state(STATE_OFF)
345.         state = 46
346.         continue
347.
348.     if (state == 46) and
(abs(normalize_angle_difference(load_enable_angle + 90 -
sputnik.navigation.get_z_axis_angle())) < navig_angle_precision):
349.         sputnik.transmitter.set_state(STATE_ON)
350.         sputnik.camera.set_state(STATE_ON)
351.         load_enable_time = sputnik.cpu.get_flight_time()
352.         load_enable_angle = sputnik.navigation.get_z_axis_angle()
353.         sputnik.camera.start_shooting()
354.         state = 47
355.         continue
356.
357.     if (state == 47) and ((sputnik.cpu.get_flight_time() -
load_enable_time) >= load_enable_interval):
358.         slot = sputnik.camera.stop_shooting()
359.         sputnik.transmitter.send_photo(slot)
360.         sputnik.camera.set_state(STATE_OFF)
361.         tracking_mode.enabled = False
362.         state = 60
363.         continue
364.
365.     if state == 60:
366.         orientation_mode.desired_angle = normalize_angle(270 -
ground_station_angle)
367.         orientation_mode.desired_angular_velocity = 0.0
368.         state = 61
369.         continue
370.
371.     if (state == 61) and not orientation_mode.active:
372.         radio_enable_angle = ground_station_angle - 5.0
373.         state = 70
374.         continue
375.
376.     if (state == 70) and
(abs(normalize_angle_difference(radio_enable_angle -
sputnik.navigation.get_z_axis_angle())) < navig_angle_precision):
377.         radio_enable_time = sputnik.cpu.get_flight_time()
378.         state = 71
379.         continue
380.
381.     if (state == 71) and ((sputnik.cpu.get_flight_time() -
```

```
radio_enable_time) >= radio_enable_interval):
   382.          sputnik.transmitter.set_state(STATE_OFF)
   383.          state = 80
   384.          break
```

2.10. Критерии определения призеров и победителей

Количество баллов, набранных при решение всех задач суммируется. Призерам второго отборочного этапа было необходимо набрать 5000 баллов (для участников из 9 класса) и 10000 (для участников из 10-11 класса). Победители первого отборочного этапа должны были набрать 33000 баллов.