Satellite project, AST 1100

5 Part 5: Launching the satellite mission

Having now written software making it possible for the satellite to orient itself, we will launch the satellite. You have (hopefully) simulated the planned trajectory and prepared in detail for every eventuality. Now your task is to ensure the actual satellite mission goes according to the plan. Before proceeding, you should now have calculated the total fuel consume expected to reach your destination and reported this number to your group teacher.

5.1 Launching the actual satellite

You now have a satisfactory simulated satellite trajectory. Launching the satellite "for real" is done using the `AST1100SolarSystem.sendSatellite` function, which takes as input a text file containing instructions.

Due to various factors (such as imprecise knowledge of planet positions, masses and even unknown objects in the solar system) and since the earlier simulation was done using the Euler-Cromer method, the actual trajectory will always deviate from your calculation. However, since we have fitted cameras to the satellite, we can use these mid-mission to orient it and compute its position relative to the planets in the system. The approach you should use, which is illustrated in Fig. 1, is the following:

(i) Launch the satellite with the same initial conditions as in your simulation.

(iii) Just before your first planned boost, orient the satellite using the software you developed in part 3 (the first time you use it, you will need the instructions given in section 5.2 at the end here). In this way you obtain the actual position and velocity of the satellite before the boost is performed.

(iv) You will see that there is a (hopefully) small deviation from your calculated satellite position and velocity at this point. You therefore need to go back to your calculations of the satellite trajectory and change the position and velocity just before the boost with the actual position and velocity. Rerun your calculation from this point on. Determine based on this whether changes are needed in the upcoming boost in order to achieve the planned close encounter with the target planet.

(v) Repeat steps (ii) to (iv) until you, hopefully, have a satellite–planet rendezvous.

Along the trajectory you should take pictures in the direction of the target planet at approporiate intervals. For the launch, note that the position you

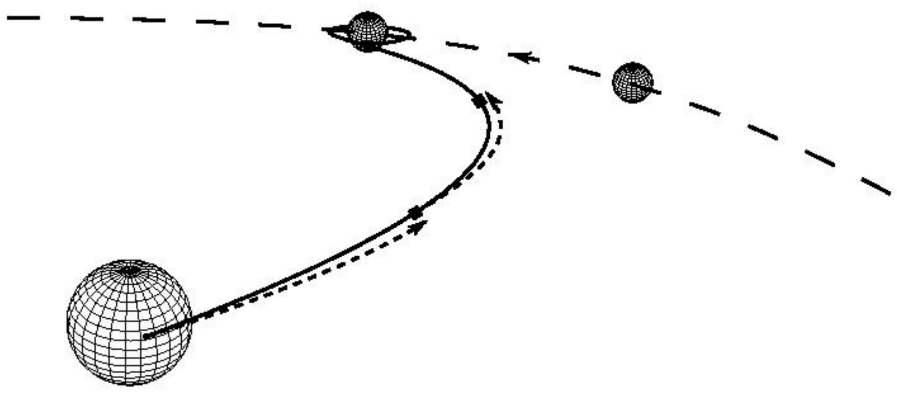

Figure 1: Simulated (dotted) and actual trajectory of the satellite. At the points in the real trajectory marked by black squares, pictures are taken and analyzed in an effort to orientate the satellite in space. Since this gives us the exact position of the satellite, we may next perform a simulation from this point on to the next time we perform the orientation operation. Please note that your trajectory and resulting orbit may look (a lot) more complicated.

specify as the launch position needs to be a point at the surface of your home planet. If you have made a mistake in calculating the position of the surface of the planet and try to launch away from the surface, the program will terminate and tell you that you are too far away, aswell as the maximum allowed distance.

Exercise 5.1.1: We denote the resolution of the satellite camera by P pixels $\times$ P pixels, and its field of view by $F^\circ \times F^\circ$. We let R denote the radius of the target planet. Show that the distance from the planet, L, must be

$$L \lesssim \frac{RP}{F}$$

in order for it to show up as more than a single pixel in the image. For our satellite camera, $P \approx 2000$, while $F = 70^\circ$. (Hint: Compute first the angular size of a single pixel in the image, then use the small-angle approximation for $\tan\theta$ to express θ in terms of R and L.)

Hints :

- The `sendSatellite` function takes as input the file name of a text-file containing instructions. The possible instructions in the text file are
  - (i) `launch` takes the following arguments: (1) A time (often this will be $t_0 = 0$, but any time is accepted); (2, 3) initial x-, and y-coordinates of the satellite; (4, 5) initial launch v_x [AU/Yr] and v_y [AU/Yr] *relative to your home planet*; (6) the total amount of fuel you have calculated [kg] for the whole trip; (7) the constant $\frac{dp}{dt}$ $[kg * m/s^2]$ of *one box* of your rocket engine, that you calculated in part 1; (8) the number of boxes used in your rocket engine; and (9) the number of particles pr. sec that leaves 1 box.

(ii) `boost` takes the following arguments: (1) The time for the boost; (2, 3) change of velocity vector, Δv_x [AU/Yr] and Δv_y [AU/Yr].

(iii) `picture` : A time (1) and direction is needed. The direction is given in terms of a θ (2) and ϕ (3) (spherical polar coordinates) and an up-direction (4, 5, 6). The up-direction is given as a 3D vector.[1]

(iv) `video` : There are 2 ways of creating a video. You can either (A) make the camera focus on a planet of your choice (it's important to choose the right planet since a planet very far away will not be visible). For this option 2 lines are needed. First [video $time_1$ planetToWatch] and on another line [video $time_2$ planetToWatch] Or you can (B) choose starting angles and end angles and make the camera turn as you please. The command for this option is [video $time_1$ θ_1 ϕ_1] and then on another line [video $time_2$ θ_1 ϕ_2]. Only one video can be created for one command file. The video will be output as an xml file to be uploaded by MCAst.

The commands in the input file must be ordered chronologically. Furthermore, at most two events can be at the exact same time (but of course you can separate two events by e.g. 10^{-7} years). An example input file might look like this (please note that this is an *example* and you will most likely have to use more precision (more significant digits) in an *actual* command file):

```
launch    0.0   2.6   0.1   0.2  0.9   4000
          5.37e-09   1e13   1.98e+14   #( one line!)
orient    0.6
boost     0.7     0.5     0.3
picture   2.5     3.14    1.57   0.0    0.0    1.0
video     3.0     1
video     3.1     1
boost     3.3    -0.1    -0.1
```

where

```
launch    0.0   2.6   0.1   0.2  0.9   4000
          5.37e-09   1e13   1.98e+14
```

means launch at $t_0 = 0\,\text{Yr}$, at initial position $\mathbf{r}_0 = (2.6, 0.1)\,\text{AU}$, with initial velocity $\mathbf{v}_0 = (0.2, 10.9)\,\text{AU/Yr}$, Total fuel = 4000kg, $\frac{dp}{dt} = 5.37 \cdot 10^{-9}$, number of boxes = 10^{13}, and number of particles pr sec = $1.98 \cdot 10^{14}$. The next line,

```
orient     0.6
```

means we are performing an orientation at $t = 0.6\,\text{Yr}$. The orientation is either a manual one, or an automatic orientation depending on whether or not you have already tested and verified your manual orientation software. The next line,

[1] Using the analogy of an airplane in flight, you can think of θ and ϕ as determining the pitch and yaw angles, while the roll angle is determined by the up-direction. See e.g. `https://en.wikipedia.org/wiki/Aircraft_principal_axes`. Often, the up direction of [0, 0, 1] - along the z axis - will be okay.

```
boost      0.7      0.5       0.3
```

means at $t = 0.7\,\mathrm{yr}$, change the velocity by $\Delta\mathbf{v} = (0.5, 0.3)\,\mathrm{AU/Yr}$. Using the same engine as defined in the launch. The next line after this,

```
picture     2.5       3.14      1.57    0.0      0.0      1.0
```

means at $t = 2.5\,\mathrm{yr}$ take a picture in direction given by $\theta = 3.14$ and $\phi = 1.57$, with the up-direction $\mathbf{r}_{\mathrm{up}} = (0.0, 0.0, 1.0)\,\mathrm{AU}$.

```
video       3.0       1
video       3.1       1
```

is the first video option where your camera i focused on planet 1. The video commands for starting and ending the video do not need to be consecutive.

```
video      2.6      3.14      1.57
video      2.7      2.14      1.57
```

is the second video option where the camera rotates between $\theta_1 = 3.14 \rightarrow \theta_2 = 2.14$ and from $\phi_1 = 1.57 \rightarrow \phi_2 = 1.57$. This might not work while not in orbit.

5.2 Testing the orientation software

The first time you use the `orient` command you have to manually do the orientation in order to make sure that your code and methods work. Once this manual orientation has been done once, subsequent orientation commands will make the satellite automatically return the position and velocity in a file (e.g "orient1.npy" for the first orientation command etc.).

Following the first orientation command, the manual testing procedure goes as follows:

- You will recieve a .png file, whick is a picture taken in the forward direction of the satellite's orientation. You will then be prompted to input the corresponding angle. Use your orientation software to find this angle.
- If you successfully found the angle, you will receive the measured $\Delta\lambda$ of the two reference stars. Use this data to estimate the velocity of the satellite. Provide these velocities when you are prompted.
- If you have found correct velocities you will be given a file with the distances to all the planets. Use this information to compute the position of the satellite and provide this when prompted.
- If all of these steps are completed successfully you have proved that your orientation software works, thus you can safely just ask the satellite for the position and velocity when you need it for simulation.

5.3 Goals of this part

Your first task is to launch the rocket successfully. Feel free to make a take-off video (direct the camera towards planet 0) starting at the launch time and ending some hours later.

Your main goal is to create a command file which takes your satellite all the way to the last boost which is the orbital injection maneuver at your destination planet. Compare the position found from orientating the satellite with the calculated position of your target planet at the time of the last boost. Does the distance seem reasonable?