Satellite project, AST 1100

Part 4: Skynet

The goal in this part is to develop software that the satellite can use to orient itself in the star system. That is, that it can find its own position, velocity as well as its orientation using the different sensors aboard the satellite itself. You will have to show that the software you develop here allows the satellite to orient itself. After this manual test has been successfully completed, the satellite will perform these operations automatically. This means that you can at any point ask the satellite for its own position and velocity. The test and how to ask the satellite for this information will be explained in the next part of the project.

4.1 Orientation

The first step is to figure out the orientation of the satellite itself. A neat way to do this is to take a picture of the sky in the forward direction and figure out where on the sky it is pointing. You have all been given a spherical picture of the whole background sky, you can now take the picture and compare it to this background. We use spherical coordinates to denote the points on the sphere, defined as in fig. 1.

Stereographic projection

In order to compare the pictures you take from the satellite to the background sky you need to make projections from the spherical sky to a flat surface to

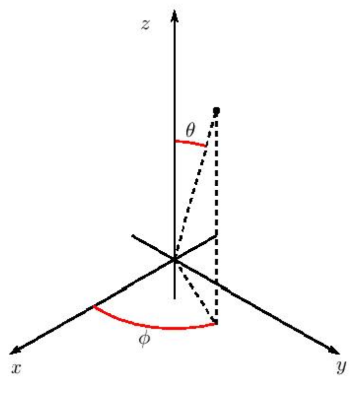

Figure 1: Definitions of the angles θ and ϕ in our spherical coordinate system.

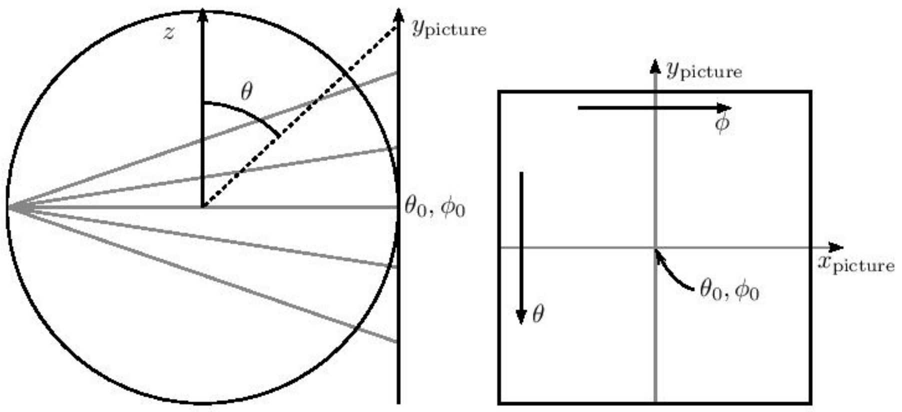

Figure 2: Sketch showing how the stereographic projection is made. On the left is shown a slice of the unit sphere and shows the lines of projection from the sphere to the tangent plane. The intersection of the straight lines with the right hand side of the circle are the points on the sphere which are projected onto the flat picture on the right hand side (of the left figure). The field of view (FOV), α, determines the maximum values of $\theta - \theta_0$ (along the y_{picture}-axis) and $\phi - \phi_0$ along the x_{picture}-axis shown in the picture, i.e. $(\theta - \theta_0)_{\max} = \alpha_\theta/2$ for $\phi = \phi_0$ and $(\phi - \phi_0)_{\max} = \alpha_\phi/2$ for $\theta = \theta_0$. Here, α_θ and α_ϕ denotes the FOV in the θ and ϕ directions, respectively.

compare it to the flat picture take by the satellite. One way to do this is to use *stereographic projection*, this maps each position on the sphere, denoted by θ and ϕ, onto a point on the tangent plane of the surface about some point (θ_0, ϕ_0). Since we are restricting ourselves to the x-y-plane (equator) we will only do projections where $\theta_0 = \pi/2$. The points on the tangent surface are denoted by the coordinates x_{picture} and y_{picture}, and (θ_0, ϕ_0) corresponds to $x_{\text{picture}} = y_{\text{picture}} = 0$. (See fig. 2)

Take care not to think of x_{picture} and y_{picture} as the coordinates of space (which are called x, y and z), remember that now these are just the coordinates on the tangent plane, defined such that the y_{picture}-axis is along the negative θ direction and the x_{picture}-axis is along the positive ϕ direction.

The projection is given by the transformation equations:

$$x_{\text{picture}} = k \sin\theta \sin(\phi - \phi_0), \tag{1}$$

$$y_{\text{picture}} = k\big(\sin\theta_0 \cos\theta - \cos\theta_0 \sin\theta \cos(\phi - \phi_0)\big). \tag{2}$$

where

$$k = \frac{2}{1 + \cos\theta_0 \cos\theta + \sin\theta_0 \sin\theta \cos(\phi - \phi_0)}. \tag{3}$$

The inverse transformations, which are the ones we need, are given by:

$$\theta = \frac{\pi}{2} - \sin^{-1}\left(\cos c \cos\theta_0 + \frac{y_{\text{picture}} \sin c \sin\theta_0}{\rho}\right), \tag{4}$$

$$\phi = \phi_0 + \tan^{-1}\left(\frac{x_{\text{picture}} \sin c}{\rho \sin\theta_0 \cos c - y_{\text{picture}} \cos\theta_0 \sin c}\right), \tag{5}$$

where

$$\rho = \sqrt{x_{\text{picture}}^2 + y_{\text{picture}}^2}, \tag{6}$$

$$c = 2\tan^{-1}\left(\frac{\rho}{2}\right). \tag{7}$$

We will be working with projections in the equatorial plane with the $z-$direction upwards. For these projections x_{picture} (the coordinate in the tangent plane) will be the horizontal coordinate of the projections (picture), while y_{picture} will be the vertical coordinate.

Optional exercise: If the camera onboard the satellite has a field of view α_ϕ in the horizontal direction, show that the the maximal and the minimal value of x in the picture is given by:

$$x_{\text{max/min}} = \pm\frac{2\sin(\alpha_\phi/2)}{1+\cos(\alpha_\phi/2)}. \tag{8}$$

Show likewise that if the field of view in the in the vertical direction is given by α_θ, we get:

$$y_{\text{max/min}} = \pm\frac{2\sin(\alpha_\theta/2)}{1+\cos(\alpha_\theta/2)}. \tag{9}$$

(Hint: For $x_{\text{max/min}}$ look at the case when $y = 0$. Likewise for $y_{\text{max/min}}$ look at the case when $x = 0$. These cases greatly simplify the trigonometric expressions in equations 1 and 2.)

Look at a sample picture (for instance sample0000.png) taken by the satellite. How many pixels does it have in the horizontal and the vertical direction? You can see this by loading the picture into a python array as explained in the next section.

Goal 1: Make 360 projections with the same numbers of pixels as the sample image, and with field of view $\alpha_\phi = \alpha_\theta = 70°$; One centered on each degree of ϕ_0 in the equatorial plane.

Make a grid of x and y values corresponding to each of the pixels in the projection. Use the inverse formulas for stereographic projection to find the values of θ and ϕ corresponding to each x and y coordinate. You then have to use the `ang2pix` function in the `AST1100SolarSystem` module to go from these θ and ϕ values to a pixel on the spherical background. Save these 360 projections as png files on your computer, or even better, as one big numpy array with shape [360, ypix, xpix, 3] which can be opened with `np.load()`. A technical detail here is that since all rgb values are integers between 0 and 255, you can save a big amount of space by using the keyword argument `dtype=np.uint8` when initializing these arrays. Now each element of the array takes up 8 bits space on memory and disc instead of 64 bits.

The numpy array file himmelkule.npy contains the RGB values of the whole celestial sphere. The `ang2pix` function takes two arguments `theta` and `phi` and returns an integer corresponding to the pixel on the spherical background. It is a static function, meaning that it can be called without making an instance of the class `AST1100SolarSystem`. This is an example of how to use the celestial sphere array together with `ang2pix`:

```
inFile = open('himmelkule.npy', 'rb')
himmelkulen = np.load(inFile)
inFile.close()
theta, phi = pi/2, 0
pixnum = AST1100SolarSystem.ang2pix(theta,phi)
temp = himmelkulen[pixnum]
rgb = [temp[2], temp[3], temp[4]]
```

The variable `rgb` now contains three numbers, corresponding to the red, green and blue value of the celestial sky in $\phi = 0$, $\theta = \pi/2$. `temp[0:1]` is not used (supposed to contain theta and phi).

PNG files

In a png-file, colors are represented by three integers from 0 to 255, these are the RGB values for each pixel.

Here is an example code that shows how you can open and manipulate png files in python:

```
from PIL import Image
import numpy as np

img = Image.open('example.png')                        #Open existing png
pixels = np.array(img)                                 #Make png into numpy array
width = len(pixels[0,:])
redpixels = [(255, 0, 0) for i in range(width)]#Make array of red pixels
pixels[500,:] = redpixels                              #Insert into line 500
img2 = Image.fromarray(pixels)
img2.save('exampleWithRedLine.png')                    #Make new png
```

As you may well see, this code opens a png file and makes a new png with a horizontal red line at row 500. Notice that images have the y-coordinate as their first index, and the x-coordinate as their second index.

Goal 2: Write code to find the orientation of the satellite given an input png. Your code should take a filename as imput and print the orientation angle ϕ_0 (in degrees) that fits best.

Each time you want to find the orientation of the satellite, you can look at a picture from the satellite (in the equatorial plane) and compare this picture to the 360 projections you made. You can do this systematically by doing a least square fit on the difference between the pictures. When doing the comparison of the pixels, do the least square fit over all the colors.

Hint: You can substract whole numpy arrays from each other as long as they have the same shape, even images consisting of [ypix, xpix, 3] values.

4.2 Velocity

The next step is to find the velocity of the satellite with respect to the star in your star system. In order to do this we can use the Doppler effect. By measuring the position of the H_α spectral line in two stars positioned at different angles, you can measure the Doppler shift and thereby the two components of your velocity vector. You will be given access to the function `getRefStars()` which gives you the shift in wavelength $\Delta\lambda$ for the H_α line at 656.3nm for these two stars with respect to your star. A positive shift means that the reference star is moving away from your home star.

Goal 3: Use information from `getRefStars()` to calculate the radial velocities of the reference stars v_{refstar} in the rest frame of your star.

The two reference stars are are found at some angles ϕ_1 and ϕ_2. By analyzing the spectra from these reference stars taken from the spectrograph on the satellite, and using the doppler effect, you will be able to obtain the radial velocity component of your satellite with respect to the reference star. When the orientation starts, the satellite will return $\Delta\lambda$ for each of the reference stars, as seen from the satellite. The radial velocity of the reference star relative to the satellite is $v_{\text{rel}} = v_{\text{refstar}} - v_{\text{sat}}$. Where v_{sat} is the velocity of the satellite in the direction of the reference star. v_{sat} in the direction ϕ_1 of reference star 1 will be called v_1, and in the direction ϕ_2 of reference star 2 will be called v_2. From this information, you should be able to find the velocity component of the satellite in the directions ϕ_1 and ϕ_2 in your regular coordinate system (the one relative to your home star).

Lets define

$$\hat{u}_1 \equiv \begin{pmatrix} \cos\phi_1 \\ \sin\phi_1 \end{pmatrix}, \tag{10}$$

and

$$\hat{u}_2 \equiv \begin{pmatrix} \cos\phi_2 \\ \sin\phi_2 \end{pmatrix}. \tag{11}$$

These are the unit vectors in the directions ϕ_1 and ϕ_2.

The velocity components that you can measure using the spectrograph + doppler effect are the following

$$v_1 = \mathbf{v}\cdot\hat{u}_1 = v_x\cos\phi_1 + v_y\sin\phi_1, \tag{12}$$

$$v_2 = \mathbf{v}\cdot\hat{u}_2 = v_x\cos\phi_2 + v_y\sin\phi_2, \tag{13}$$

these are the radial velocity components in the directions of the two reference stars. These equations can be written on matrix form

$$\begin{pmatrix} v_1 \\ v_2 \end{pmatrix} = \begin{pmatrix} \cos\phi_1 & \sin\phi_1 \\ \cos\phi_2 & \sin\phi_2 \end{pmatrix} \begin{pmatrix} v_x \\ v_y \end{pmatrix}. \tag{14}$$

This is fine, but we have v_1 and v_2 and want to find v_x and v_y, so we need the inverse relationship

$$\begin{pmatrix} v_x \\ v_y \end{pmatrix} = \frac{1}{\sin(\phi_2-\phi_1)} \begin{pmatrix} \sin\phi_2 & -\sin\phi_1 \\ -\cos\phi_2 & \cos\phi_1 \end{pmatrix} \begin{pmatrix} v_1 \\ v_2 \end{pmatrix}. \tag{15}$$

Goal 4: Make a program that can take values of $\Delta\lambda$ of the Hα line from each of the two reference stars, and calculate the velocity (v_x, v_y) of the satellite in the coordinate system with your home star at the origin. Check that your program gives you what you expect if both $\Delta\lambda$ are the same as in the rest frame of the star (then the satellite has zero velocity), and if e.g. both $\Delta\lambda$ are zero (your satellite has the same velocity as each of the stars along their given direction).

Optional exercise: (For those who like linear algebra.) Show that the inverse of the matrix

$$\begin{pmatrix} \cos\phi_1 & \sin\phi_1 \\ \cos\phi_2 & \sin\phi_2 \end{pmatrix}$$

is given by

$$\frac{1}{\sin(\phi_2 - \phi_1)} \begin{pmatrix} \sin\phi_2 & -\sin\phi_1 \\ -\cos\phi_2 & \cos\phi_1 \end{pmatrix}.$$

Optional exercise: For what values of ϕ_1 and ϕ_2 does this inverse not exist? What does these special values mean for about $\hat{u}_1$ and $\hat{u}_2$? Are $\hat{u}_1$ and $\hat{u}_2$ linearly independent?

4.3 Position

We will assume that the satellite can use radar to find the distance from itself to either of the planets. This is not an entirely trivial exercise and you will not do it, we will simply assume that the satellite itself can figure out the distance to all of the different planets (and the star) at any time you want. The satellite will return a list containing all these distances in AU `[ dist_p0, dist_p1, dist_p2, ..., dist_star ]`.

What you have to do is use the list of distances to all of the planets to obtain the position of your satellite (remember that you have already calculated the positions of all the planets at a list of times and can use interpolation to get the positions at any given time). How to do this is up to you. We will discuss some approaches in class.

Goal 5: Write a program that takes as input a list containing the distances to all planets and the star, as well as the current time of the measurement.

Hint: To test your implementation of Goal 5, you should be able to generate a random x and y position within your solar system at some time $0 < t < 20$, and generate the list of distances from this position to each of the planets and the sun, `[ dist_p0, dist_p1, dist_p2, ..., dist_star ]`. Run this list through your analysis to check if you retrieve the correct random coordinate you started with. Try this a couple of times.