AST 2000 - Part 4
Onboard Orientation Software

Welcome to Part 4 of the AST2000 Spacecraft Project. The aim of this part is to develop navigation software for your spacecraft.

GOALS

∇ Write software that determines the angular orientation of the spacecraft.

∇ Write software that analyses the velocity of the spacecraft using data from its onboard equipment.

∇ Write software that analyses the position of the spacecraft using data from its onboard equipment.

RELEVANT MATHEMATICS

The following section includes relevant mathematics that you need in order to solve this part's challenges. You will not be asked to explain this section in the same level of detail as presented here. Your job is to understand what is needed in order to solve your specific challenge.

1. The Spherical Coordinate System

You have likely come across the spherical coordinate system in earlier courses, but let's go through a brief recap of the essentials. There are several ways to define your spherical coordinate system; we will adapt the convention shown in figure 1. Using this convention, the spherical coordinate ranges are:

$$0 \leq r \leq \infty, \quad 0 \leq \theta \leq \pi, \quad 0 \leq \phi \leq 2\pi \tag{1}$$

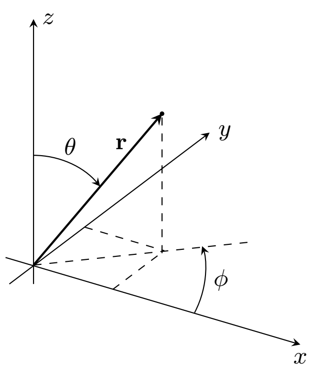

FIG. 1. The spherical coordinate system, defined according to the convention presented in (1).

2. Stereographic Projection

1. Coordinate Transformations

A stereographic projection is a map that transforms coordinates on the surface of a sphere to coordinates on a plane. For example, figure 2 shows a stereographic projection of Earth's surface.

FIG. 2. A stereographic projection of Earth. This image was created by manipulating a texture map from http://www.shadedrelief.com/natural3/pages/textures.html

There are several conventions for stereographic projections but we are specifically going to use the convention shown in figures 3 and 4. Figure 3 shows a two-dimensional slice of the three-dimensional surface shown in figure 4.

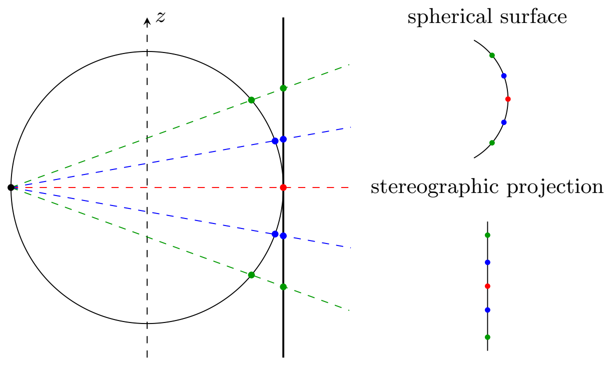

FIG. 3. An illustration of the principles behind stereographic projections. The left-hand side shows the geometric process while the right-hand side depicts the original surface and the resulting stereographic projection separately.

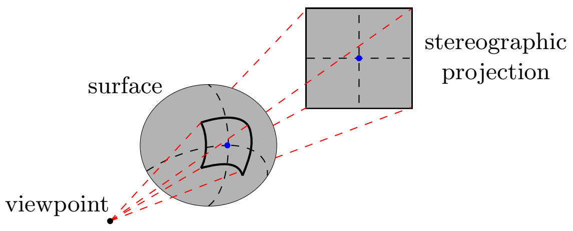

FIG. 4. A three-dimensional illustration of stereographic projections. The transformation of a straight line on the spherical surface to stereographic projection is illustrated in figure 3.

Mathematically speaking, the stereographic projection relates spherical coordinates (θ, ϕ) to planar coordinates (X, Y). As illustrated in figure 4, the stereographic projection is dependent on its viewpoint. This is handled by letting the center of the stereographic projection (i.e., where $X = Y = 0$), transform a specific spherical coordinate (θ_0, ϕ_0). The relationship between (θ, ϕ) and (X, Y) is exemplified in figure 5. The so-called *coordinate transformations* are the mathematical functions that relate (θ, ϕ) to (X, Y) and vice versa.

$$X = \kappa \sin\theta \sin(\phi - \phi_0) \tag{2a}$$
$$Y = \kappa\big(\sin\theta_0 \cos\theta - \cos\theta_0 \sin\theta \cos(\phi - \phi_0)\big) \tag{2b}$$

$$\theta = \theta_0 - \arcsin\left[\cos\beta\cos\theta_0 + \frac{Y}{\rho}\sin\beta\sin\theta_0\right] \tag{3a}$$
$$\phi = \phi_0 + \arctan\left[\frac{X\sin\beta}{\rho\sin\theta_0\cos\beta - Y\cos\theta_0\sin\beta}\right] \tag{3b}$$

where

$$\frac{2}{\kappa} = 1 + \cos\theta_0\cos\theta + \sin\theta_0\sin\theta\cos(\phi - \phi_0) \tag{4a}$$
$$\rho = \sqrt{X^2 + Y^2} \tag{4b}$$
$$\beta = 2\arctan\left(\frac{\rho}{2}\right) \tag{4c}$$

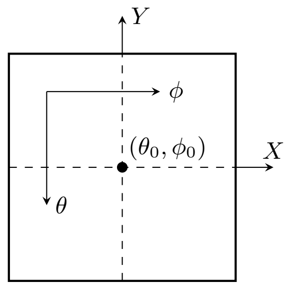

FIG. 5. The planar coordinate system of a stereographic projection centered about the spherical coordinate (θ_0, ϕ_0).

2. *Pictures*

We are now going to take stereographic projection one step further and look at pictures. [1] A normal camera is not able to project the entire spherical surface onto the picture, the boundaries of the projection is dependent on the camera's *field of view* (or FOV).

FOV, denoted by α, is defined as the maximum angular width of the picture:

$$\alpha_\theta = \theta_{\max} - \theta_{\min}, \quad \alpha_\phi = \phi_{\max} - \phi_{\min} \tag{5}$$

This introduces new coordinate ranges on θ and ϕ:

$$-\frac{\alpha_\theta}{2} \le \theta - \theta_0 \le \frac{\alpha_\theta}{2}, \quad -\frac{\alpha_\phi}{2} \le \phi - \phi_0 \le \frac{\alpha_\phi}{2} \tag{6}$$

With limitations on θ and ϕ, the stereographic projection introduces equivalent limitations on X and Y:

$$X_{\max/\min} = \pm\frac{2\sin\left(\alpha_\phi/2\right)}{1 + \cos\left(\alpha_\phi/2\right)} \tag{7a}$$
$$Y_{\max/\min} = \pm\frac{2\sin\left(\alpha_\theta/2\right)}{1 + \cos\left(\alpha_\theta/2\right)} \tag{7b}$$

3. **A Little Linear Algebra**

We are now going to take a look at a two-dimensional surface. By now you should be (somewhat) comfortable describing a plane using either (x, y) rectangular coordinates or (r, θ) polar coordinates, both of which have many useful properties. However, these are not the only two ways of describing a two-dimensional surface. In this section we are going to look at a third way of describing a two-dimensional surface, which in reality is actually a generalization of the standard (x, y) coordinate system.

It is common to represent the Cartesian coordinate vectors $\hat{x}$ and $\hat{y}$ in the following way:

$$\hat{x} = \begin{pmatrix}1\\0\end{pmatrix} \quad \text{and} \quad \hat{y} = \begin{pmatrix}0\\1\end{pmatrix} \tag{8}$$

Using these coordinate vectors you can describe any 2-dimensional vector $\mathbf{d}$ as a *linear combination*:

$$\mathbf{d} = \begin{pmatrix}d_x\\d_y\end{pmatrix} = d_x\hat{x} + d_y\hat{y} = (\mathbf{d}\cdot\hat{x})\hat{x} + (\mathbf{d}\cdot\hat{y})\hat{y} \tag{9}$$

However, as you will learn in MAT1120, describing an arbitrary 2-dimensional vector as a linear combination is not something that is unique to $\hat{x}$ and $\hat{y}$. As long as you have 2 unit vectors in the same plane, say $\hat{u}_1$ and $\hat{u}_2$, you are essentially good to go:

$$\mathbf{d} = (\mathbf{d}\cdot\hat{u}_1)\hat{u}_1 + (\mathbf{d}\cdot\hat{u}_2)\hat{u}_2 \tag{10}$$

But what are $\hat{u}_1$ and $\hat{u}_2$? They form what is known as *a basis for* $\mathbb{R}^2$, you will study the details in MAT1120. Our interest lies in the connection between (u_1, u_2) and (x, y).

Let us quantify the relationship between (u_1, u_2) and (x, y) using figure 6: The unit vectors $\hat{u}_1$ and $\hat{u}_2$ can be viewed as $\hat{x}$, rotated about the origin by angles ϕ_1 and ϕ_2. Note that $\phi_2 < 0$ in figure 6.

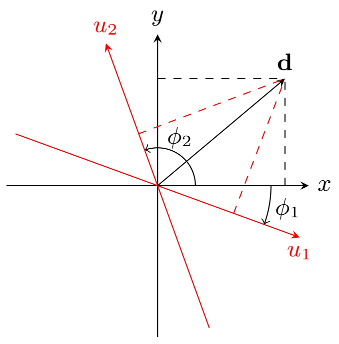

FIG. 6. Changing coordinate systems from (x, y) (black) to (u_1, u_2) (red). The vector **d** must therefore change to appropriate coordinates.

Using figure 6, you should be able to derive the following relations between the two coordinate systems:

$$\hat{u}_1 \equiv \begin{pmatrix} \cos\phi_1 \\ \sin\phi_1 \end{pmatrix} \quad \text{and} \quad \hat{u}_2 \equiv \begin{pmatrix} \cos\phi_2 \\ \sin\phi_2 \end{pmatrix} \tag{11}$$

This allows us to explore $d_1 \equiv \mathbf{d} \cdot \hat{u}_1$ and $d_2 \equiv \mathbf{d} \cdot \hat{u}_2$:

$$\begin{aligned} d_1 &= d_x \cos\phi_1 + d_y \sin\phi_1 \\ d_2 &= d_x \cos\phi_2 + d_y \sin\phi_2 \end{aligned}$$

or in matrix form:

$$\begin{pmatrix} d_1 \\ d_2 \end{pmatrix} = \begin{pmatrix} \cos\phi_1 & \sin\phi_1 \\ \cos\phi_2 & \sin\phi_2 \end{pmatrix} \begin{pmatrix} d_x \\ d_y \end{pmatrix} \tag{12}$$

What is equation (12)? Well, it transforms the vector **d** from the (x, y) plane to the (u_1, u_2) plane. Note that it is misleading to denote (x, y) and (u_1, u_2) as separate planes as this is actually not the case. They both describe the same plane, each using a different set of coordinate vectors. If you want to transform **d** from the (u_1, u_2) plane to the (x, y) plane, the equivalent transformation is

$$\begin{pmatrix} d_x \\ d_y \end{pmatrix} = \frac{1}{\sin(\phi_2 - \phi_1)} \begin{pmatrix} \sin\phi_2 & -\sin\phi_1 \\ -\cos\phi_2 & \cos\phi_1 \end{pmatrix} \begin{pmatrix} d_1 \\ d_2 \end{pmatrix} \tag{13}$$

If you're having a difficult time understanding the coordinate transformations, try to set $\phi_2 = \phi_1 + 90°$. The result is the "normal" rotation matrix you may have encountered before. Our coordinate transformation is actually a generalization of the rotation matrix.

4. RGB images and PNG files

Before embarking on the challenges, you need to learn handling PNG-files.

In a png-file, colors are represented by three integers from 0 to 255 for each pixel. These are the RGB (Red-Green-Blue) values. The color of the pixel is a combination of these three colors weighted by each of these integers. As an example red is represented by $(255, 0, 0)$, white would be $(255, 255, 255)$.

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

```python
from PIL import Image
import numpy as np

img = Image.open('example.png') # Open existing png
pixels = np.array(img)                           # png into numpy array
width = len(pixels[0, :])
redpixs = [(255, 0, 0) for i in range(width)] # Array of red pixels
pixels[500, :] = redpixs                          # 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.

CHALLENGES

In order to perform the correct boosts, your spacecraft needs to be able to orient itself. The orientation includes the rotational orientation of the satellite, its current velocity and its current position.

A. Generating Reference Pictures

The stars in the night sky are very distant compared to the distances within your solar system. We are therefore going to assume that the night sky remains constant over the course of this project. Knowing that the night sky is constant allows us to generate a library of reference pictures that cover the entire celestial sphere. Later when the satellite takes a picture of the night sky, you will be able to compare this picture to your reference pictures and finally deduce the rotational orientation of your spacecraft.

To generate the reference pictures we are going to use a brilliant satellite that is orbiting your home planet right now! You can access data from this satellite using the NumPy array file `himmelkule.npy`, which contains a pixelized spherical RGB image of the sky. Each (θ, ϕ) coordinate on the celestial sphere is mapped to a specific pixel index using the static method `get_sky_image_pixel` of the `SpaceMission` class. For each pixel there are three integers, red, green and blue. Note that the `himmelkule.npy` array actually has 5 integers for each pixel, the first two are not in use, you obtain the R, G and B values from index 2, 3 and 4.

1. The Pixel Grid

This first point is optional for those working alone: Use the stereographic projection transformations in order to show equations (7). 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.

2. Generating Pictures

Let us start by generating a flat picture from a part of the spherical picture. For comparison with your projection you will use the file `sample0000.png` which is a stereographic projection with FOV $\alpha_\theta = \alpha_\phi = 70°$ centered at the position $\phi = 0°$ and $\theta = 90°$. Note that $\theta = 90°$ corresponds to the plane of your solar system.

1. Open the picture `sample0000.png` using the `PIL` module and determine the size of the picture in pixels.

2. Find the full range of the coordinates (X, Y) in the picture.

3. In order to try to reproduce this picture making a stereographic projection form the sphere, generate an (X, Y) grid, then use this to generate the corresponding (θ, ϕ) grid.

4. Use your coordinate grids, `get_sky_image_pixel`, and `himmelkule.npy` in order to generate the full RGB picture centered about $\phi = 0°$ and $\theta = 90°$. The picture you generate is supposed to be the same as `sample0000.png`, use this as a reference to see if you have generated the picture correctly.

3. Generating 360 Pictures

This point is not compulsory, it depends on how you choose to solve the next challenge. But in most cases this will become very useful: Use the same method and generate 360 flat pictures, each picture centered about $\phi = i$, where $i = 0°, 1°, 2°, \ldots, 359°$. Do not forget to save the data! You only need to generate the reference data once. These pictures may be useful in the next step.

B. Image Analysis

With your reference pictures handy, your next task is to write a general function that determines the angle ϕ_{new} that a new picture is most likely centered about. The function structure should be similar to this:

input	output
`png` picture	ϕ_{new}

There are several ways of comparing two images, one possibility is using a least squares approach. In this way you are essentially measuring the difference between the `RGB` values of the two images.

C. Doppler Shift Analysis

In Part 2 of the project you analysed the radial velocity of a star in order to study the planets in its solar system. You are now going to use the radial velocity of two stars in order to study your own spacecraft's velocity.

The wavelength of the H_α spectral line is 656.3 nm as seen from a rest frame. Assuming your spacecraft has used your software from challenge B in order to determine its rotational orientation, it is now able to rotate and point its equipment towards any known star in the night sky. Using onboard equipment, your spacecraft is able to measure the Doppler shift $\Delta\lambda$ in the H_α spectral line.

You need two stars in order to resolve the x and y components of your spacecraft's velocity with respect to your sun. The reference stars have been pre-determined by a team of excellent astronomers from your home planet. You can obtain the ϕ coordinates, ϕ_1 and ϕ_2, of the two stars from the `star_direction_angles` attribute of your `SpaceMission` instance. Additionally, the Doppler shifts of the two stars as seen from your sun are available in the `star_doppler_shifts_at_sun` attribute.

1. The Process

1. Find a formula that converts wavelength data into radial velocity with respect to a reference star.

2. Use your formula to determine the radial velocities of your sun with respect to the reference stars.

3. Assume for the time being that your spacecraft's spectrograph measures $\Delta\lambda = 0$ for both radial velocities. Use your formula again and determine the radial velocities of your spacecraft with respect to the reference stars. Then find the (ϕ_1, ϕ_2) velocity components of your spacecraft with respect to your sun.

4. Transform your (ϕ_1, ϕ_2) velocity into (x, y) velocity.

2. The General Case

Rewrite your code for C 1 into a function that accepts the Doppler shift of the reference stars (as measured by your spacecraft's spectrograph) as an input. The structure should look something like this:

Here, v_x and v_y are the velocity components of your spacecraft with respect to your sun (the "normal" velocity).

`input`	`output`
$\Delta\lambda$ from the star at ϕ_1	v_x
$\Delta\lambda$ from the star at ϕ_2	v_y

To test your function, think of a situation in which you know both the spacecraft's velocity and the corresponding $\Delta\lambda$ values.

D. Spacecraft Trilateration

Trilateration is the process by which one determines the position of a body using measured distances to known positions.

In this challenge, you are going to do exactly this. Your task is to write a function that determines your spacecraft's position based on a list of distances from your spacecraft to the planets and the sun, measured at a specific time.

Using an onboard radar array, your spacecraft is capable of measuring the distance between itself and another object. Fortunately, the radar is installed with automagic planet-recognition software, meaning it will include the radii of the planets and sun into its calculations. The distances measured by the radar are from the spacecraft to the center of the planets/the sun.

The function structure should look something like this:

`input`	`output`
Time of measurements	x
List of measured distances	y

Here, x and y are the position components of your spacecraft with respect to your sun.

Also, the list of measured distances is given in AU and will be structured in the following way:

$$\begin{matrix}\text{List of}\\ \text{measured distances}\end{matrix} = \begin{bmatrix}\text{dist to} & \cdots & \text{dist to} & \text{dist to}\\ \text{planet } 0 & & \text{planet } n & \text{sun}\end{bmatrix}$$

This challenge is not entirely trivial and you have not been given much to start with. It is up to you to design a numerical algorithm. Try to come up with a rigorous way of determining the position to any given degree of accuracy. Meanwhile, try to reduce the number of computations as much as possible by using the information given to you.

MANUAL ORIENTATION

Congratulations, you have now completed Part 4! The final step is to verify that your software is accurate, you are therefore going to complete a manual orientation:

1. The first step is to use the satellite's onboard equipement in order to gather the necessary data. This is done using the following methods of your `SpaceMission` instance: `take_picture` (rotational orientation), `measure_star_doppler_shifts` (velocity) and `measure_distances` (position).

2. The next step is analyse the data using your orientation software.

3. The final step is to use the `verify_manual_orientation` method in order to verify your calculations have been done correctly.

Further details on how these methods work can be found in the documentation of the `ast2000tools` package. Note that the manual orientation begins at the exact moment you finish the launch, i.e. when you are *in space*.

AUTOMAGIC ORIENTATION

After completing the manual orientation your software will be uploaded to the satellite, which allows it to orientate itself automatically. What this means is that you only have to perform the manual orientation once! Following the manual orientation, you can easily run the software at any point during the trip with a simple command.

EXTRA CHALLENGE

If you love linear algebra, this one is for you. [2]

1. You may have noticed that equations (12) and (13) are linear transformations. Let

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

and find A^{-1}.

2. Provided you want to describe a two-dimensional plane, can you find a reasonable argument for why $\hat{u}_1$ and $\hat{u}_2$ cannot be parallel? Are $\hat{u}_1$ and $\hat{u}_2$ linearly dependent?

FIG. 7. Orbitally oriented, trajectory-planning, motivational duck.

AST 2000 - Part 4
Tips, Hints & Guiding Questions

Do not be frightened by the mathematics in Part 4! You have been equipped with all the necessary theory. In addition to the mathematics and problem solving skills you will learn in this part, our intention with Part 4 is for you to realise how large the field of Astronomy really is. *In astrophysics, you need to know a little bit of everything in order to do a little bit of anything.*

Remember that the coordinates for each picture's center should be:

$$X = Y = 0$$

which is equivalent to:

$$\theta_0 = \frac{\pi}{2},\ \phi_0 = i = 0^\circ, 1^\circ, \ldots, 359^\circ$$

[1] Technically, there's not much else to do with pictures.
[2] Personally, I find it to be very straightforward.