A Graphical Interface for Creating Hops to the Stellar Objects

Akshay E[1], Chaithanya Naik[2,6], Harshda Saxena[2,6], Liza Dahiya[2], Pratham Patil[2,6], Sahyadri Krishna[3], Vedant Shenoy[2,6], Vivek Reddy Pininti[4] and Yasha Kaushal[5]

[1]Central University of Karnataka, Kadaganchi 585367, Kalaburagi, India
[2]Indian Institute of Technology Bombay, Powai, Mumbai 400076, India
[3]Indian Institute of Science Education and Research, Tirupati 517507, India
[4]Osmania University, Hyderabad 500007, India
[5]University of Pittsburgh, Pittsburgh, PA 15260, USA
[6]Krittika-The Astronomy Club of IIT Bombay, Powai, Mumbai 400076, India

Project with Krittika - The Astronomy Club of IIT-B

This document contains the alternate design of the Graphical Interface, including the description for the algorithms I implemented to select and store the hops to stellar objects. All the sections described here are my alternate approach to the one in parent repository we created. The stellar data along with the interface for viewing the hops can be found in the parent repository. The parent repository has the complete interface.

Here, the entire graphical user interface was designed in Python. K-Dimensional tree was implemented for efficiently selecting the stars based on their magnitude with user clicks. The selected hops are then stored in a way such that no user intervention is required outside the interface. All the functionality of this interface that removes the redundancy while creating hops across several sessions, while also enhancing the user experience are illustrated in the USER MANUAL in this repository.

- Vivek Reddy Pininti

Contents

Chapter 1

The Design

1.1 Visualising the Data

The purpose of hopping is to reach an object of lower magnitude with the aid of objects of higher magnitude which are located in the neighborhood[1] of it. The editor interface allows a user to create hops for objects of interest, which in the case of the current iteration of this interface is the 110 messiers of the messier catalogue. This interface can be scaled-up to accommodate other databases as well, for example, hops for NGC objects[2] have been tried as well during the internal versions of this interface.

To create hops we need marker stars which would be the stars that direct us towards the object of interest. We would want the marker stars to be non-messier objects, essentially stars, which in this case are from the Tycho-1 catalogue.

The messiers contain a variety of objects which can been classified into globular clusters; open clusters; galaxies; nebula, supernova remnant; and others of which include associations of stars, composite objects and active

[1] neighborhood is related to the projection on celestial sphere and not the actual distances between the objects

[2] The NGC database has messiers too which would be a problem when plotting both the messier catalogue and the NGC; especially that the messiers in both messier and NGC catalogues don't have the same right ascension, declination values even for the same epoch J2000. This issue is solved by additionally adding NGC designations to all the messiers in datatable of messier catalogue and marking all the messiers in the NGC table matching the NGC designations. This information can be used to plot data from both the messier and NGC tables by omitting duplicate messiers in the NGC table.

galactic nuclei. Objects belonging to each of these classifications are represented using different marker shapes in the plot. Besides, all the objects in Tycho-1 catalogue are stars, which in the interface are represented with the same color and shape.

When we look at stars in the night sky, one distinguishing feature is how bright one object is compared to others. Betelguese, one of the brightest stars in the night sky got a lot of attention last year, in part because of a sudden drop in it's magnitude leading to the discussion of whether it's going to go supernova anytime soon. Though it would make for a spectacular show in the sky, it turned out the dimming wasn't intrinsic and was because of a dust cloud[1]. The brightness in astronomy is a unit less measure called magnitude. The scale of the magnitude is a reverse logarithmic. It means an object with negative magnitude is brighter than the one with positive magnitude; every interval of one magnitude equates to a variation in brightness of $\sqrt[5]{100}$ or roughly 2.512 times. The magnitude data we would be using is the apparent magnitude i.e., the magnitude of an object as observed from earth.

This apparent magnitude has to be visualised on the plot containing objects of discrete magnitudes. That can be achieved by allowing sizes of objects to depend on the magnitude of the object. For that the following relation has been used:
A magnitude difference $m_1 - m_2 = \Delta m$ implies a brightness factor of

$$\frac{F_2}{F_1} = 100^{\frac{\Delta m}{5}}$$

From the above relation, marker size for the scatter plot is chosen to be

$$\frac{constant}{(\sqrt[5]{100})^V}$$

constant is decided as fit for the plot keeping in mind the overcrowding of scatter points and the size of the lowest magnitude object; while V is the visual magnitude of the object.

Once the size has been chosen, it is fixed[3] even when the graph is narrowed down by zooming-in. So at two different field of views of the plot, a magnitude

[3]This behavior has been observed with the matplotlib default plotting. This could be overcome by using other libraries for plotting but each has their own advantages and drawbacks

M star would be of same size. This gives rise to a problem which is that the object of lowest magnitudes would be almost non visible and user would wonder if there's a star in the chart or if it's a speck of dust. The feature given highest priority for a star to be a marker star is its visual magnitude. So if they are too small as much as not being visible, it makes one's job of creating hops easier as those of lower magnitudes are disregarded anyway. But the messiers for which we are intending to hop towards would be non visible too which is not how we would want it to be. One way to solve this is to choose a different constant in the above relation for sizes of messiers but it would mean the messier and the stars from Tycho-1 wouldn't be linearly related in terms of visual magnitude. This issue has been resolved by adding buttons to the interface for decreasing and increasing sizes of messier objects, which at the lowest value would mean the same constant for both messiers and tycho-1 stars. This allows one to identify messiers of lowest magnitude too which otherwise would be too small.

Now, all the objects from both the databases are plotted in a 2D Right Ascension vs Declination plot with right ascension and declination on x- and y- axes respectively, using the matplotlib library[2]. This would not be a true representation of what one would see in the sky, especially when the the entire sky is visualized at once. But at a narrow fields of view, the localised plot would fairly be adequate for creating hops.

Earlier, it was mentioned that the messiers have been classified for representing them on the plot. The symbols have been chosen so as to mimic an IAU sky chart. But the standard matplotlib markers will not make the cut as is. All except for galaxies, the data has been plotted twice with different markers, sizes, colors when needed but for the galaxies an ellipse SVG has been created using open source software Inkscape to use as a marker. The reason for why SVGs haven't been used for other required markers as well instead of plotting the same data twice for other objects is the inability to use multi colors when using SVG paths.

1.2 Ways of Selecting Marker Stars

All the visualization discussed until now is essentially to aid an user in choosing marker stars which would in turn aid the user to hop to a messier object during a field observation through a non-automated telescope.

A star is selected to be a marker based on its proximity to the object

of interest and its visual magnitude. A user will be able to estimate the relative magnitudes with the help of marker sizes which depend on the visual magnitude data and can also read the exact value by hovering over an object; the axis limits of the plot at any level indicate the distance between objects on the celestial sphere.

One way to do this is to allow the user to click on objects only. In this method, the clicks in an empty region of stars would not be recorded and a click on an object would record the data of that object and add it to the hop sequence. The drawback here is that the user has to precisely click on an object for it to be added to the hop sequence and this drawback is enhanced to different lengths at different fields of view. So, this method can be discarded. Another way to do it is to enable users to click anywhere on the plot and add to the hop sequence, the brightest among stars within a *certain distance* from the point of click. This *certain distance* can be made a fixed quantity.

Overall, this method of selecting marker stars from user clicks is a better way among the discussed two. Further, a naive way to implement the latter process is to calculate distance to all stars in the given database from the point of click. Then identify the stars that fall within a *certain distance* from the point of click, essentially considering only those stars that lie inside a circle with center as point of click and radius of the *certain distance*. Then the brightest of those stars is to be identified and added to the hop sequence. This would mean that the calculation of distances from point of click for all the objects in the database has to be repeated every time the user clicks to select a star. For a small dataset, this could have satisfied the purpose but as the dataset becomes large, it becomes infeasible. As an example, the Tycho-1 catalogue we would be using for selection of marker stars for hopping contains around 133110 stars with magnitudes lower(brighter) than 9.0.

1.3 The Algorithm

1.3.1 The Selection

Instead of using brute force to calculate neighbours within a given proximity, a kd-tree is constructed using scipy[3].

The `scipy.spatial.KDTree` function constructs a k-dimensional tree also known as kd-tree. It essentially is a binary tree. The kd-tree has an

advantage over using brute force and essentially reduces the time complexity from $\mathbf{O[DN^2]}$ to $\mathbf{O[logN]}$ once the tree is constructed for the entire dataset which by the way will be only done once the editor interface is run. This method is very fast for dimensions less than 20 and in our case the data is 2 dimensional.

The function for returning the data of objects within a certain distance will take two arguments; one is the *certain distance*, the other is the coordinates for the point of click by the user. This step would return a very small subset of stars from the original large dataset and then the coordinates of the star with highest magnitude is added to the hop sequence.

Another requirement here is regarding how the data of the click is sent to the tree. This is done using the native event handling of matplotlib which allows us to call any function based on an event on the plot generated, which in this case would be the mouse clicks by the user.

1.3.2 The Storing

In the end there must be a way to store the data of all the hops. Here we have to store the coordinates of stars added to the hop sequence along with the details of the object to which the hops are used for. So, we would want to map the messier object to all the coordinates of marker stars in a sequence. First thing in mind when the need is clearly specified was to store the data using a dictionary. Dictionary is the simplest of ways to store data such as what's generated in case of this project. It is the most minimalistic way to sort because the keys in the dictionary could be the IDs of the messier objects or any other unique ID in case the interface is scaled-up to create hops for non-messier objects as well. Each key then would be paired to a value which is a list of lists, where the sub-list would contain right ascension and declination of the marker star and index-0 and index-1 of each sub list would contain Right Ascension and Declination of marker stars.

Now the dictionary with information must be passed on to a different program which displays the hops in sequence for a given messier. The way it is done is by pickling[4] the dictionary and loading it into the other program which would then have all the information for displaying hops for all the objects whose information is in the keys of the dictionary. A script in the program carries out the operation of loading the pickled dictionary of hops if one exists from previous sessions of using the program to create hops or else it creates a new dictionary and pickles it once hops are added for an object.

This method was implemented as a solution for the cluttering caused by another way of storing hops which was to store hops for each of the messier as .txt files in the working directory and then convert them to a csv file, especially while visualizing data using mpld3, which is a library combining matplotlib with javascript.

1.4 The Interface

[4] All the data visualised using `matplotlib` library is to be integrated into GUI for which `tkinter`[5] package has been utilised. The purpose of having a graphical interface is to provide interaction between the user and the plot by tapping into the potential of the widgets.

The interface before the first release has been through stages of improvement, which at each stage had updates ranging from improvements in aesthetics to altogether newly added features.

The algorithm section took care of describing how the essentials of the program works. Now, it is about the features in the graphical interface that make the job of creating hops easier and prettier.

First of all, about navigating the plot; on starting up the interface, the entire stretch of sky is visible with all messiers and the stars from tycho-1 plotted with their sizes as a function of their visual magnitudes, though the messier are too small one cannot identify more than one. One can zoom in and out of the plot, pan across by using the default navigation toolbar's hand tool and rectangular zoom tool of matplotlib. Besides, one can narrow down to a messier using the name resolving search bar, using either common name or the messier ID. As mentioned in the plotting section, two buttons for changing messier sizes are present. One can get the details of an object by hovering over it. For this, mplcursors has been used. A check button to turn on/off the display of constellation stick figures is also present. Reset button is used to revert the plot to what it was on opening the interface, clearing out other navigational changes along with the display constellation borders and also the change in messier sizes.

Now, to the hopping; a button which displays the IDs of messiers for which hops have already been created is present. To start creating hops one has to enter the ID to which hops are to be created and as the stars

[4] In depth details about each widget of the interface can be found in the user manual in the github repository

are selected by mouse clicks or deselected by using undo button, they are updated in real-time in the region above plot. On finishing creating hops for a given object of interest, a finish hop button would then store the hops. The testing phase put light on bugs related to aesthetics as well as functionalities, which were corrected as soon as they were reported. At this stage, the stage has been set for the first release. Otherwise, one can get all fancy with the GUI forgetting all about releasing the interface for general use.

1.5 The Interface for Viewing the Hops

The interface basically displays hops to an object of interest. The hops are expected to have been created using the other interface.

The interface would ideally need the details of the object of interest, which would be provided using the name resolving search bar; along with the fields of view of the eyepiece and the finderscope for generating two plots, one with a view through telescope and the other with a wider view through finder scope. Besides, the aperture of the telescope is also required which would then be used to calculate the limiting visual magnitude. This helps in figuring out whether the object of interest would be observable through the given telescope.

If the user doesn't provide the details of FoVs, the plot can be based on fixed default values for the same. But if the aperture is skipped, the plot can return hops for the object of interest irrespective of the magnitude.

The following relation has been used to calculate the limiting visual magnitude[6] for a telescope with aperture of D in mm:

$$\text{limiting } m_v = 2 + 5 \log_{10} \text{D}$$

If the magnification of the eyepiece along with the apparent field of view of the eyepiece, true field of view can be calculated from the following relation:

$$\text{True FoV} = \frac{\text{Eyepiece Apparent FoV}}{\text{Magnification of the Eyepiece}}$$

Validating an Object of Interest

All good until now until a user enters an object of interest for hops without any information related to whether the object would be in sky field at the time, date, location of observation.

For this information, user can optionally enter the details of observation. Then, the coordinates for object on successfully getting resolved by the search field would be calculated by passing the string of resolved object's name or ID to `astropy`'s[7] `SkyCoord.from_name()`

Later on, the information of time, date, location is used to transform the right ascension and declination to alt-azimuth using
`obj_coord.transform_to(AltAz())`

Among altitude and azimuth, only azimuth is considered to provide to the user, the information regarding whether the object of interest would be above or below the horizon at the intended point of observation.

References

[1] E. M. Levesque and P. Massey, "Betelgeuse just isn't that cool: Effective temperature alone cannot explain the recent dimming of betelgeuse," 2020.

[2] J. D. Hunter, "Matplotlib: A 2d graphics environment," *Computing in science & engineering*, vol. 9, no. 3, pp. 90–95, 2007.

[3] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. Jarrod Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. Carey, İ. Polat, Y. Feng, E. W. Moore, J. Vand erPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and S. . . Contributors, "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python," *Nature Methods*, 2020.

[4] G. Van Rossum, *The Python Library Reference, release 3.8.2*. Python Software Foundation, 2020.

[5] F. Lundh, "An introduction to tkinter," *URL: www. pythonware. com/library/tkinter/introduction/index. htm*, 1999.

[6] *BAA Handbook*. British Astronomical Association, 1999.

[7] A. M. Price-Whelan, B. Sipőcz, H. Günther, P. Lim, S. Crawford, S. Conseil, D. Shupe, M. Craig, N. Dencheva, A. Ginsburg, *et al.*, "The astropy project: Building an open-science project and status of the v2. 0 core package," *The Astronomical Journal*, vol. 156, no. 3, p. 123, 2018.