Astronomy

January 20, 2021

1 Identification of Celestial Bodies

Our night sky is on a cloudless night is one littered with light. These lights are celestial bodies extremely far away that could be distant stars, or even galaxies. Though despite the vast number of celestial bodies that make them seem endless and identical, some bodies still have some distinct qualities to them that help us identify them when viewed from a different location/time.

One of these formations of stars or asterism that is part of the the constellation, Ursa Major is the Big Dipper. The Big Dipper is made up of bright, identifiable, stars that when viewed, looks like a dipper, spoon, ladle. We only know these stars because they were positioned in a shape that made them identifiable when observed in relation with every other star of the asterism. That is the basic idea behind constellations as well. These groups of stars only received wide recognition due to the fact that they created recognizable shapes when observed as a group.

This is what we are trying to achieve without the help of an observable visual pattern. By taking a patch of night sky that includes all the celestial bodies you would see in that patch, we will try to devise an algorithm that when given a smaller patch of our patch, we can identify the celestial bodies regardless if the patch has been transformed, rotated, or dilated in anyway.

We would start off with obtaining our patch of sky. We try to get a mostly uniform patch of sky and we would do this by first deciding on what part of the sky we would want. When describing locations on Earth, we use longitude and latitude while when we are describing locations in the sky, we would use right ascension and declination. In our case, we pick our right ascension and declination to range from 180 degrees to 181.62 degrees and 43.85 degrees to 45 degrees respectively.

We then would use these position parameters and make a query to the Sloan Digital Sky Survery database in order to get information on the celestial bodies inside our patch of sky.

[93]:
```
#Code provided by Dr. John Ringland
#import the libraries that we need
import requests
from io import StringIO
import pandas as pd

#Our query string
q = '''SELECT TOP 100000
   p.objid,p.ra,p.dec,p.u,p.g,p.r,p.i,p.z,
   s.specobjid, s.class, s.z as redshift
FROM PhotoObj AS p
   JOIN SpecObj AS s ON s.bestobjid = p.objid
```

```
WHERE
   p.ra BETWEEN 180 AND 181.62 and
   p.dec BETWEEN 43.85 AND 45
'''
#Sloan Digital Sky Survey website
url = 'http://skyserver.sdss.org/dr14/SkyServerWS/SearchTools/SqlSearch?
 ↪cmd={}&format=csv'.format(q.replace(" ","%20"))
r = requests.get(url)

#If we successfully get a response from the server, we take the returned CSV␣
 ↪file and read it into a dataframe
if r.status_code==200:
    csv = StringIO(r.text)
    df = pd.read_csv(csv,skiprows=1)
    display(df.head())
else:
    print("unsuccessful")
```

```
                objid          ra        dec         u         g         r  \
0  1237661872939073866  180.474860  44.912173  22.65754  21.29497  20.37337
1  1237661850394166115  180.105016  44.454457  24.89294  23.83855  23.36518
2  1237661872402203235  180.638896  44.533391  22.40390  20.91432  19.37721
3  1237661850931036788  180.195379  44.859022  24.79819  21.98236  20.36770
4  1237661850931167390  180.538095  44.877138  26.30246  21.03177  19.24720

          i         z            specobjid   class  redshift
0  20.26740  20.14058  7477226183001612288    STAR -0.000734
1  19.88595  18.42799  1541401554513324032    STAR -0.000313
2  18.59760  18.08331  1541401004757510144  GALAXY  0.293244
3  19.55420  19.23905  7478483749643984896  GALAXY  0.488021
4  18.53533  18.21984  7478495019638169600  GALAXY  0.408716
```

Now that we have a database with the coordinates of the celestial bodies in our patch, we just need to give each body a nickname to distinguish them and to project them onto the tangent plane so we can have a x and y value for each celestial body.

[197]:
```
#Code by Dr. John Ringland
import matplotlib.pyplot as plt
import numpy as np

df['brightness'] = (df['g'].max()-df['g'])/(df['g'].max()-df['g'].min()) ␣
 ↪#makes dimmest=0 and brightest=1
dfb = df[ df['brightness']>.7 ].copy()  # select just the brightest objects

#Do the projection on to the tangent plane
dec0 = 44.5  # representative declination for the patch
dfb['x'] = dfb['ra']/np.cos(dec0*np.pi/180) #creating the x coordinate value
```

```
dfb['y'] = dfb['dec'] #creating the y coordinate value
```

This code creates a copy of our initial dataframe and modifies it by adding x and y values to each celestial body.

[198]:
```
namelist = pd.read_csv('yob2018.txt',header=None)[0].values
dfb['nickname']= namelist[np.random.
 ↪choice(list(range(len(namelist))),len(dfb),replace=False)]
sortedDF = dfb.sort_values(by=['brightness'],ascending = False)
```

Now using this code, we added nicknames to the dataframe to distinguish each celestial body more easily.

[199]:
```
workingDF = sortedDF[["x","y","nickname"]]
# We do this because all we really need from the dataframe at this point is the␣
 ↪x,y,and nickname
```

[200]:
```
workingDF.head()
```

[200]:

	x	y	nickname
361	252.944834	44.006547	Everlea
216	252.590789	44.406619	Jasher
182	253.342142	44.237294	Keturah
253	252.862853	44.370059	Yaritzi
25	253.364462	44.089933	Yashua

We can see that we now have a sorted by brightness dataframe with x,y,coordinates,and a nickname.

We now need a method in order to classify each of our celestial bodies. Just like the stars in a constellation, they are meaningless and indistinct by themselves but when viewed in context with other closer stars, they start to become distinguishable. We will then need to classify each star relating to two other stars.

We want to consider the points in terms of a triangle where each celestial body is a vertex of a triangle. We can do this by taking taking triples of three points and creating a unique triple for every combination of points in the patch.

[313]:
```
def listOfTriples(df):
    xs = df["x"].tolist() #writes the column of the dataframe to a list
    ys = df["y"].tolist()
    listofXY = []
    for i in range(len(xs)):
        coordinates = (xs[i],ys[i]) #creates a triple of x and y values to use␣
 ↪as coordinates
        listofXY.append(coordinates)
    return listofXY
```

This function made a list of 2 item lists that contains the x and y coordinates but now we need to create a 3 item list for each combination of points.

[202]:
```
def triangles():
    listOfCombinations = []
    trips = listOfTriples()
    for i in range(len(trips)):
```

```
        for j in range(i+1,len(trips)):
            for k in range(j+1,len(trips)):
                listOfCombinations.append([trips[i],trips[j],trips[k]])#grabs␣
↪unique triples of every coordinate pair in triples
    return listOfCombinations
```

Now that we have successfully created our list of unique combinations of coordinates, we now essentially have a list of triangles.

To distinguish each triangle and to be able to pinpoint it back to our known points, we need to establish a value to each triangle to identify it.

We can standardize this triangle by having the longest side always be positioned horizontally such that one of the points of the longest side is at the origin and the other is on the x-axis with the distance between them being the length of the longest side of the triangle but we will later standardize this triangle further by normalizing the sides of the triangle by the length of the longest side.

Through the use of trigonometry, we will pick the point that is not on the x-axis after normalizing the triangle to be the represented point of the triangle.

The first step in doing this is to determine a consistent direction we want to be using for all triangles.

[203]:
```
def isCounterClockwise(triple):
    x0 = triple[0][0]
    x1 = triple[1][0]
    x2 = triple[2][0]
    y0 = triple[0][1]
    y1 = triple[1][1]
    y2 = triple[2][1]
    area = ((x0+x1)*(y1-y0)+(x1+x2)*(y2-y1)+(x2 + x0)*(y0-y2))/2 #formula for␣
↪area derived from Stokes Theorem
    if area > 0:
        return triple

    else:
        triple[1],triple[2] = triple[2],triple[1] #swap the order of the points
        return triple
```

We choose for the relative direction of the triangle to be clockwise. So that the side clockwise from the longest is the side opposite point B, which is one of the points of the longest side of the triangle resting on the x-axis. The function above does that for us by swapping the order of the triples and giving us the correct orientation.

[204]:
```
import math

def betsyP(triple):
    working = isCounterClockwise(triple)
    A = working[0] #point A
    B = working[1] #point B
    C = working[2] #point C
```

```
    a = math.sqrt((C[0]-B[0])**2+(C[1]-B[1])**2) #line BC
    b = math.sqrt((A[0]-C[0])**2+(A[1]-C[1])**2) #line AC
    c = math.sqrt((B[0]-A[0])**2+(B[1]-A[1])**2) #line AB
    longest = max(a,b,c)

    if (a == longest):
        ap = 1
        bp = b/a
        cp = c/a

        x = ((cp**2) + (ap**2) - (bp**2))/2
        y = abs(math.sqrt((cp**2) - (x**2)))
        return (x,y)
    if (b == longest):
        ap = a/b
        bp = 1
        cp = c/b

        x = ((ap**2)+(bp**2)-(cp**2))/2
        y = abs(math.sqrt((ap**2)-(x**2)))
        return (x,y)
    if (c == longest):
        ap = a/c
        bp = b/c
        cp = 1

        x = ((bp**2)+(cp**2)-(ap**2))/2
        y = abs(math.sqrt((bp**2)-(x**2)))
        return (x,y)
```

Now that we have our algorithm to return distinguished points determined by the coordinates of three points, we can now apply it to our triangle points.

[278]:
```
def returnName(coordinate):
    xx = coordinate[0]
    yy = coordinate[1]
    zz = workingDF.loc[(workingDF['x'] == xx)].iloc[0]["nickname"] #searches␣
 ↪the dataframe
    #for the coordinates and returns the nickname associated with that point
    return zz
```

[287]:
```
v = triangles()
```

[298]:
```
triangleDict = {}
for item in v:
    triangleDict[betsyP(item)] =␣
 ↪[returnName(item[0]),returnName(item[1]),returnName(item[2])]
```

We now have a dictionary with our distinguishing points acting as keys for us to access the names of the points of the triangles, one entry for every unique combination of x and y coordinates.

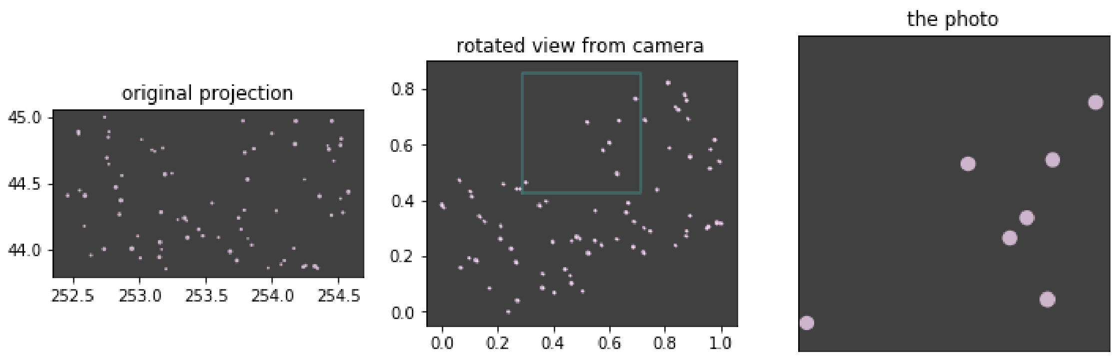
[379]:
```
#Code provided by Dr. John Ringland
def take_photo(starprojection_in):

    def random(lo,hi):
        return lo + (hi-lo)*np.random.rand()

    starprojection = starprojection_in.copy()
    starprojection['brightness'] -= starprojection['brightness'].min()
    starprojection['brightness'] += .1
    starprojection['brightness'] /= starprojection['brightness'].max()

    sp = starprojection[['x','y']].values
    plt.figure(figsize=(12,4))
    plt.subplot(131,aspect=1,facecolor='#404040')
    #plt.plot(sp[:,0],sp[:,1],'o',color='#ffddff',markersize=1,alpha=0.7)
    plt.scatter(sp[:,0],sp[:,1],c='#ffddff', s=10*starprojection['brightness'],␣
↪alpha=0.75, linewidth=0 )
    plt.title('original projection')

    sp -= sp.min(axis=0)
    sp /= sp.max()
    # data now lies in unit square

    theta = 2*np.pi*np.random.rand()  # choose a rotation angle
    c,s = np.cos(theta),np.sin(theta)
    rotation = np.array([[c,-s],[s,c]])
    rsp = (np.dot(rotation,sp.T)).T
    # scale to unit square again
    rsp -= rsp.min(axis=0)
    rsp /= rsp.max()

    w = random(.2,.6)
    xlo = random(0,1-w)
    xhi = xlo + w
    h = random(.15,.5)
    ylo = random(0,1-h)
    yhi = ylo + h
    plt.subplot(132,aspect=1,facecolor='#404040')
    plt.plot(rsp[:,0],rsp[:,1],'o',color='#ffddff',markersize=1,alpha=0.7)
    plt.scatter(rsp[:,0],rsp[:,1],c='#ffddff',␣
↪s=10*starprojection['brightness'], alpha=0.75, linewidth=0 )
    box = np.array([[xlo,xhi,xhi,xlo,xlo],[ylo,ylo,yhi,yhi,ylo]])
    plt.plot(*box,'#40a0a0',alpha=0.5)
    plt.title('rotated view from camera')

    starprojection['xprime'] = rsp[:,0]
```

```
    starprojection['yprime'] = rsp[:,1]

    photo = starprojection[ (starprojection['xprime']>=xlo) & \
                            (starprojection['xprime']<=xhi) & \
                            (starprojection['yprime']>=ylo) & \
                            (starprojection['yprime']<=yhi) ]
    #ibox = np.dot(np.linalg.inv(rotation),box)
    plt.subplot(133,aspect=1,facecolor='#404040')
    #plt.
↪plot(photo['xprime'],photo['yprime'],'o',color='#ffddff',markersize=2,alpha=0.
↪7)
    plt.scatter(photo['xprime'],photo['yprime'],c='#ffddff',␣
↪s=100*starprojection['brightness'], alpha=0.75, linewidth=0 )
    plt.xlim(xlo,xhi)
    plt.ylim(ylo,yhi)
    plt.xticks([])
    plt.yticks([])
    plt.title('the photo')
    #plt.axis('off')
    photo = photo.rename(columns={'nickname':'nickname for checking␣
↪answers','brightness':'brightness for plotting only'})
    return photo[['xprime','yprime','nickname for checking answers','brightness␣
↪for plotting only']].reset_index(drop=True)#,'brightness']]
```

With this code, when running it with our data-frame, we can get a transformed set of x and y coordinates for the celestial bodies in a portion of our patch. Though the bodies are transformed, the relational position of the celestial bodies are still held and through our algorithm we should be able to figure out which celestial bodies are in this transformed portion.

[381]:
```
example = take_photo(sortedDF)
example.head()
```

[381]:
```
     xprime    yprime nickname for checking answers  \
0  0.629262  0.497374                       Kameran
1  0.694801  0.765416                        Andria
2  0.601289  0.608364                        Adalia
3  0.578063  0.581012                          Pius
4  0.521253  0.681715                         Emari

   brightness for plotting only
0                      0.818968
1                      0.768076
2                      0.548289
3                      0.539558
4                      0.495120
```



This shows a dataframe of the augmented x and y coordinates. We will do what we did with our original dataframe and create unique combinations of triples of coordinates, which we will find the distinguishing coordinate for, then compare back to our dictionary to find the names of the celestial bodies.

[389]:
```
def listOfTriplesK():
    xs = example["xprime"].tolist() #writes the column of the dataframe to a list
    ys = example["yprime"].tolist()
    listofXY = []
    for i in range(len(xs)):
        coordinates = (xs[i],ys[i]) #creates a triple of x and y values to use as coordinates
        listofXY.append(coordinates)
    return listofXY

fff = listOfTriplesK()

def trianglesK():
    listOfCombinations = []
    for i in range(len(fff)):
        for j in range(i+1,len(fff)):
            for k in range(j+1,len(fff)):
                listOfCombinations.append([fff[i],fff[j],fff[k]])#grabs unique triples of every coordinate pair in triples
    return listOfCombinations
```

[391]:
```
uuu = trianglesK()
```

Because the augmented coordinates could not exactly match the keys of the dictionary, we need a different method to find the approximation or closest distinguishing point that is closest to the distinguishing points of the augmented coordiantes.