Research of Existing Solutions:

A. ArXiv: Repository of electronic preprints approved for posting after moderation, but not full peer review. It consists of scientific papers in the fields of mathematics, physics, astronomy, electrical engineering, computer science, quantitative biology, statistics, mathematical finance and economics, which can be accessed online., Following paper is a newly developed streak detection algorithm: https://arxiv.org/pdf/1806.04204.pdf

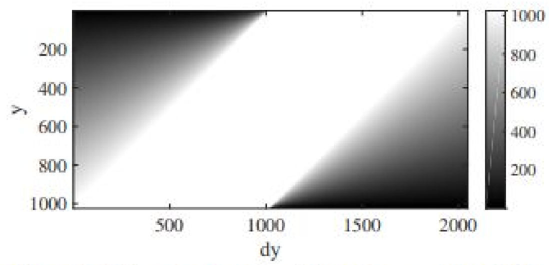


Figure 5. Radon transform of a uniform variance map for a 1024 × 1024 pixel image. Different areas of Radon space have different weight corresponding to different lengths of streaks. Dividing by the square root of this image normalizes each point in the Radon image. If the overall noise variance is not unit valued, the Radon variance map can be scaled linearly by the variance value. If the image variance is not uniform, the specific variance map should be Radon-transformed and the result used instead of a uniform Radon map.

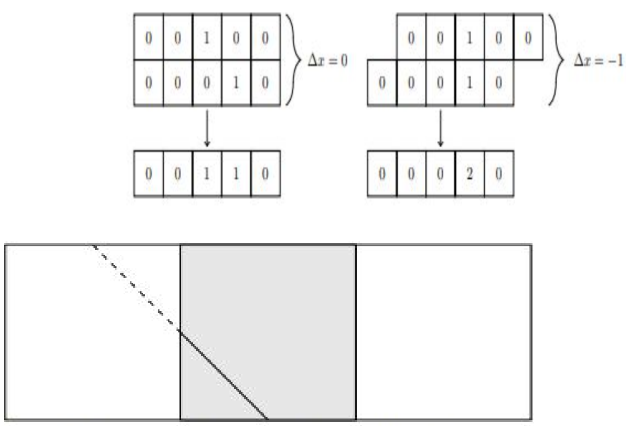
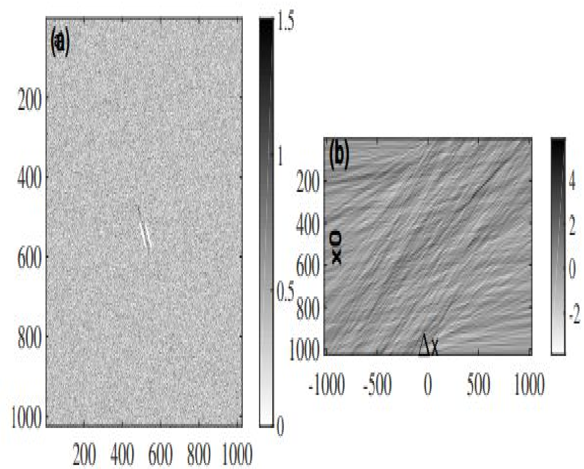
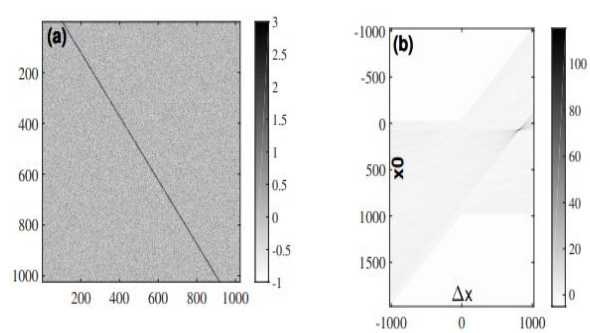


Figure 4. A cartoon of an image and zero padding. The gray area is the original image, with the solid line representing the measured streak. Since the line begins outside the image, it falls outside the Radon transform of the original image. The white areas are zero padding required so that the streak, and its continuation, marked by a dashed line, would be within the resulting Radon image. This padding of the passive axis is done in addition to padding the active axis to be an integer power of 2.



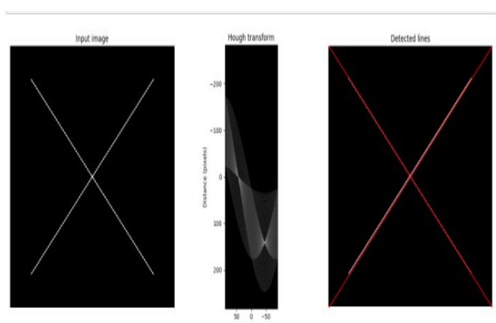


B. - Postdoctoral researcher, Dr. Ed Lin: He has worked to develop a large portion of the pipeline for image reduction of DEEP data, so he can help me identify how I should implement the algorithm. Has implemented the Hough transform in different applications

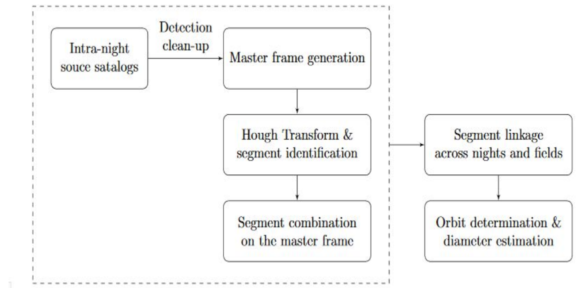
- Principal Investigator, UM Chair of Physics, Prof. David Gerdes: Discussed the problem with me and provided me with the resources necessary to pursue the project (Database access, Supplementary programs etc.). Other members of his group are working on similar work

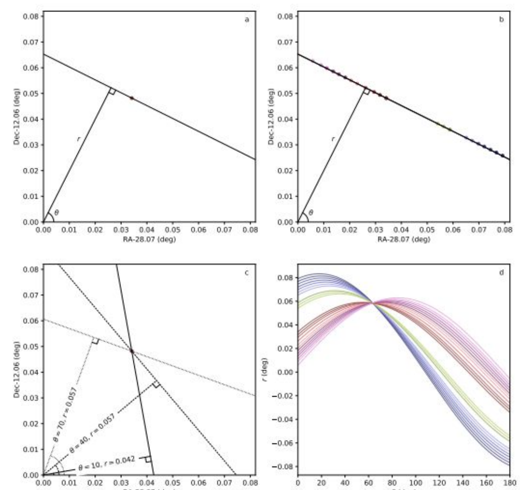
C. Information about the Hough Line Transform: https://scikit-image.org/docs/dev/auto_examples/edges/plot_line_hough_transform.html

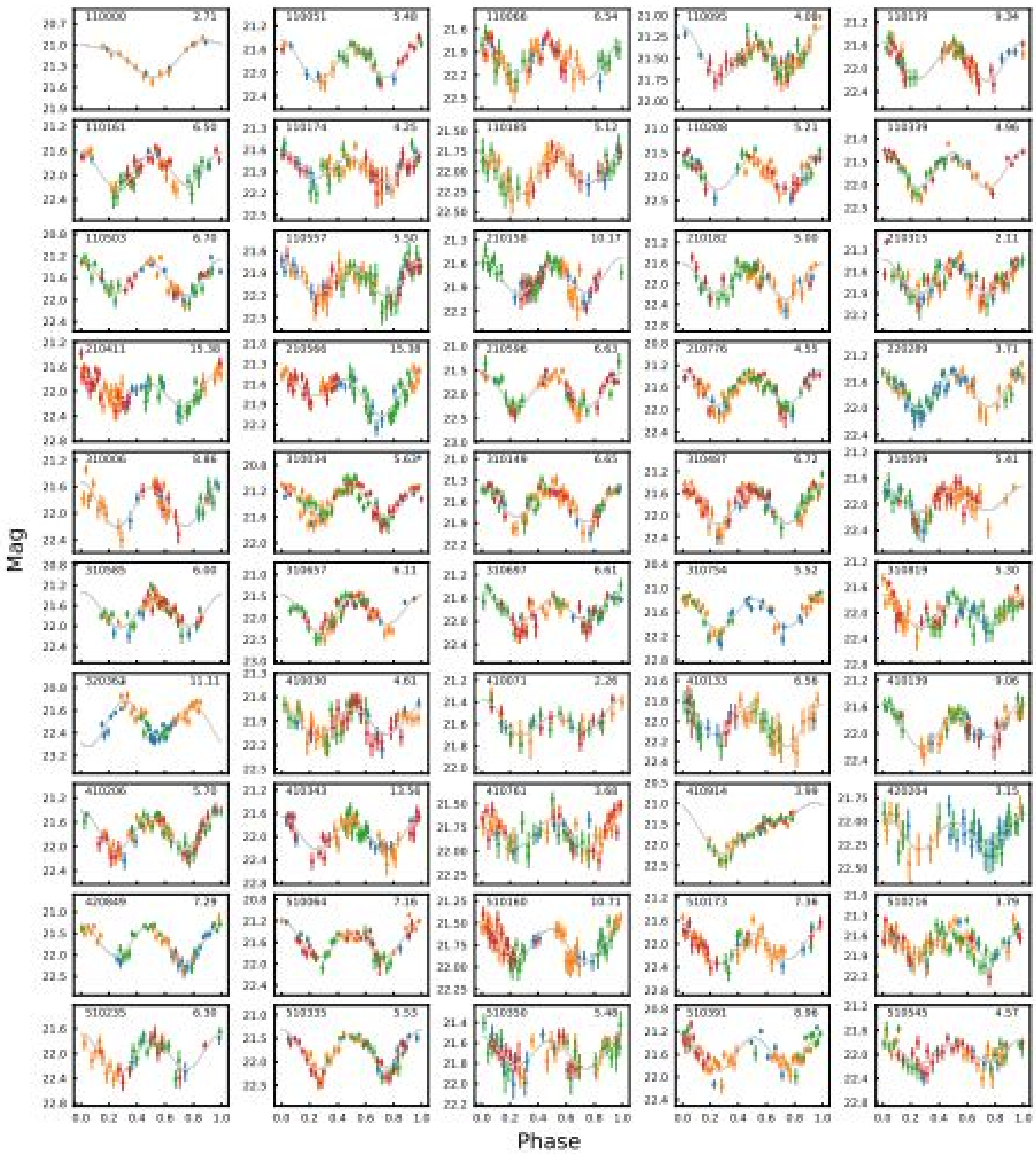
Relevant Figures:

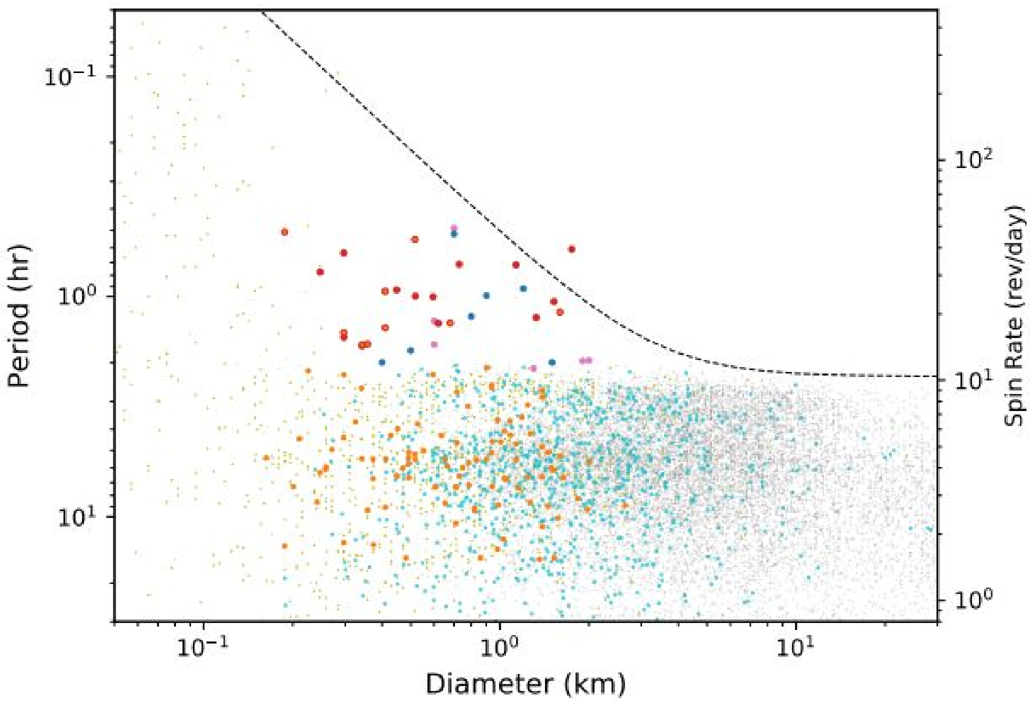


D. "Asteroid Discovery and Light Curve Extraction Using the Hough Transform -- A Rotation Period Study for Sub-Kilometer Main-Belt Asteroids" -- This technique is quite similar to the techniques I am planning to implement. One of the co-authors of this paper is a UMTNO group member. However, the surveys analyzed are entirely different. The DEEP survey is set up entirely differently and observes a much deeper field. Link: https://arxiv.org/abs/1910.07146



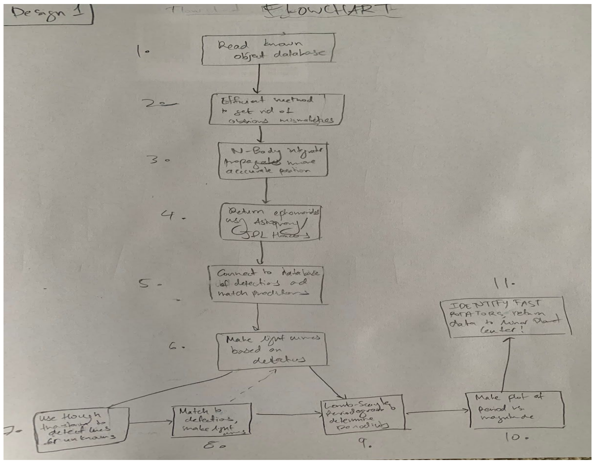



Mag
Phase



Period (hr)
Spin Rate (rev/day)
Diameter (km)
10^{-1}
10^{0}
10^{1}
10^{2}

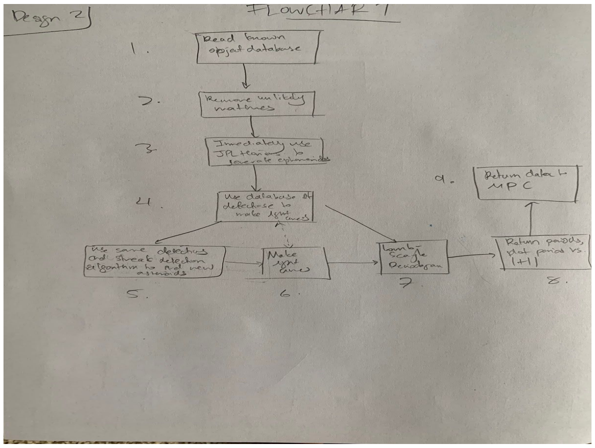
Brainstorming



In this design, the known asteroids are initially matched and the light curves constructed. This is done by applying an efficient method of discarding obvious mismatches (asteroids that are very far away from the exposure center at the time of the exposure). Then, an N-body integrator is used to narrow these positions down even further. Finally, Astroquery's JPL Horizons functionality is called to obtain 3 sigma RA and DEC uncertainties and generate ephemeride positions of the objects. Then, these ephemeride positions are compared to the actual database of detections and an array of detections is constructed. These detections are used to generate the light curves. The second part of this set of programs does the following: 1) apply Hough transform to veto stationary objects 2) generate light curves as before and 3) apply Lomb-Scargle periodogram to determine periodicity and construct plots of magnitude (size) vs. period (rotational frequency)

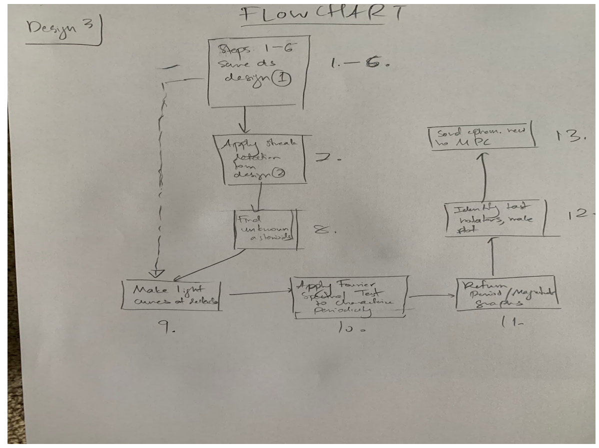
Top and bottom layers are the same as before. However, there are some modifications to the active layer in this design. Instead of TiO2 we utilize ZnO, which is much more readily available (cheaper) and is easy to synthesize even if we aren't able to acquire this from a chemical vendor. This design also has a different berry-derived dye, using a blueberry and blackberry combination. Bandgap correspondence of this dye is better suited to visible and low-UV light

Design 2



This design is quite similar to Design 1, with a couple of key changes. Steps 1 and 2 remain the same as before. However, in Step 3, we immediately call to JPL Horizons instead of passing through an N-body integration scheme. The light curve generation process is also similar. Another change arises in the unknown asteroid detection process. Instead of applying the Hough transform or the probabilistic Hough transform, the streak detection algorithm cited above is utilized (needs to be adapted to function with sparsely scattered data). The remainder of the asteroid processing is largely identical, except for the returning of unknown asteroid data to the Minor Planet Center (MPC).

Design 3



This final design integrates aspects of designs 1 and 2 in the same set of programs. The process of known asteroid detection and characterization is exactly the same as design 1, whereas the unknown asteroid detection uses the streak detection algorithm from design 2. A major change in this program is the use of the Fourier Spectral Test instead of the Lomb-Scargle periodogram. This is proposed since the spacing between consecutive detections is almost regularly sampled. In this case, the FST would provide a more accurate prediction of asteroid periodicity. However, for smaller asteroid with fewer detections, this may not be guaranteed so a technique that works well with irregularly sampled data is necessary (like the LS periodogram).

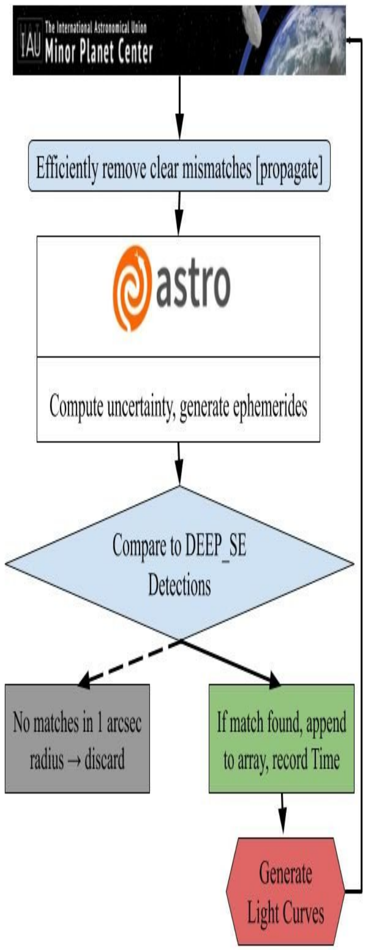
Design Constraints:

- Cost, Materials Needed: Doesn't require any additional purchases beyond the set of programs developed, implemented modules, and access to a database of detections. Will allow any researcher to find asteroids or other solar objects traveling in approximately straight lines. Could allow for crowdsourced detection to minimize computational times.
- Size: Total file size of all programs (along with generated ephemeride positions, saved images of all plots) is less than 250 MB, allowing files to be downloaded/transferred quickly (this is per set of exposures)
- Time: Per CCD (a region of 0.05 square degrees in the sky), hope to conduct all analysis and generate all plots in under 5 hours of computational time (running on my personal computer). With a quad-core, 4.00 GHz device running the loops in parallel, this could be reduced 5-fold. There are 62 CCDs in the DECam field, allowing the entire field to be analyzed in a week!
- Match efficiency, number of matches: Hope to achieve an X % identification for objects with well-defined streaks, Y% for poorer streaks ("well-defined" "poor" have not been quantified -- dependent on detections on a line, using an accumulator). Anticipate roughly 400 known asteroids in A0b field for a certain night, roughly 3-4x as many unknown asteroids

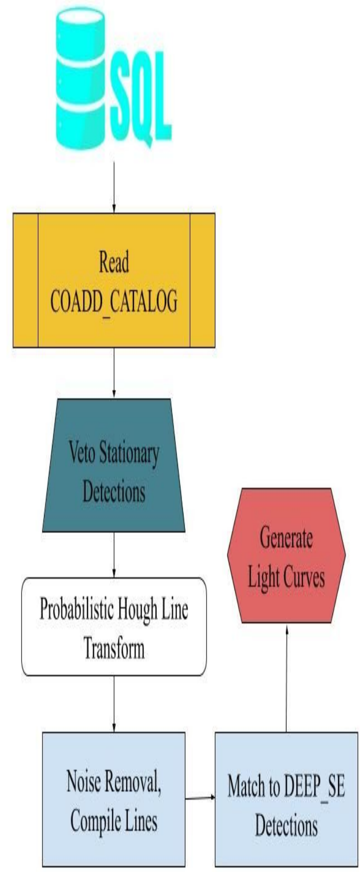
--Flowcharts on next page--

Flowcharts:

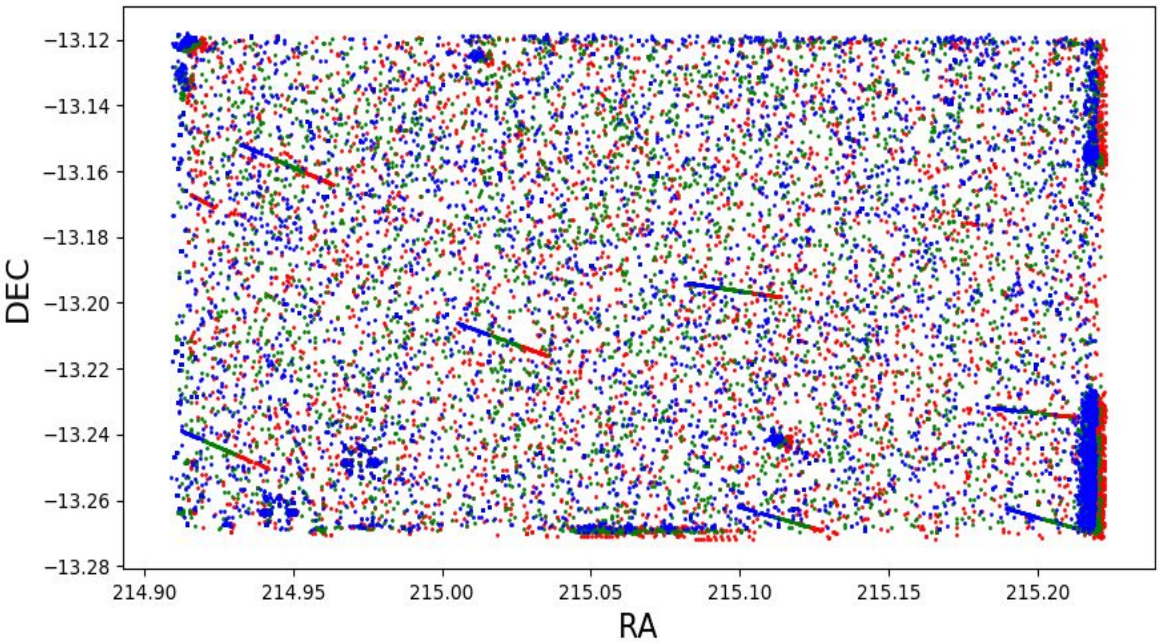
Known Asteroid Recovery:



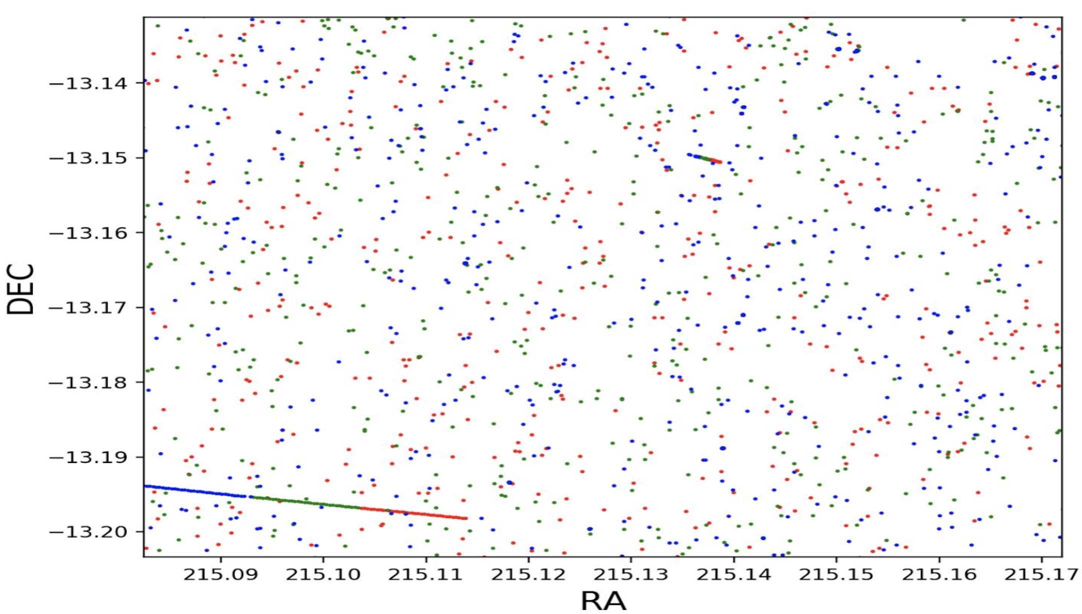
Unknown Asteroid Recovery:



Visualization of Source Catalogs Utilized: *(Red = 1st ⅓ of exposures, Blue = 2nd ⅓ of exposures, Green = 3rd ⅓ of exposures)*



Zoomed-in (Center: RA = 215.13, DEC = 13.17) → Small streak is TNO



Matches of Unknown Asteroids:

0	MPC Object Name	Principal Designation	Exposure	Exposure	RA of Obj	RA Horizo	3 Sigma U	DEC of Ob	DEC Horiz	3 Sigma U	Exposure	Exposure	DES Chip	DES Chip
270	K16U85J	2016 UJ85	845639	58576.32	214.6677	226.6358	14715.08	-13.5983	-17.2419	4165.732	215.8658	-13.6063	None	-99
271	13841	1999 XO32	845639	58576.32	215.702	215.7029	0.092	-13.1941	-13.1948	0.068	215.8658	-13.6063	S16	15
272	V4397	2005 UJ199	845639	58576.32	216.5175	216.5179	0.175	-12.6434	-12.6437	0.139	215.8658	-13.6063	None	-99
273	i9932	2015 OM24	845639	58576.32	215.5101	215.5108	0.192	-13.481	-13.4816	0.165	215.8658	-13.6063	S3	27
274	K07Te4N	Unknow	845639	58576.32	216.5492	Unknown	Unknown	-13.2507	Unknown	Unknown	215.8658	-13.6063	S19	18
275	O9215	2008 EO19	845639	58576.32	215.5862	215.5866	0.101	-12.7097	-12.7101	0.065	215.8658	-13.6063	S29	1
276	K08S92O	2008 SO92	845639	58576.32	215.3058	215.3064	2265263	-14.3148	-14.3152	762396.9	215.8658	-13.6063	N25	56
277	M7218	2005 RU3	845639	58576.32	215.5929	215.5936	0.142	-14.5718	-14.5723	0.114	215.8658	-13.6063	N29	60
278	D0046	1999 VF159	845639	58576.32	216.5054	216.5059	0.098	-13.296	-13.2965	0.089	215.8658	-13.6063	S13	24
279	A5239	2000 PD21	845639	58576.32	215.2621	215.2625	0.085	-13.0331	-13.0335	0.069	215.8658	-13.6063	S20	8
280	U8429	2005 SP141	845639	58576.32	216.6011	216.6016	0.134	-13.9815	-13.9818	0.089	215.8658	-13.6063	N19	50

1999 XO32 = Blankenship (has very well-defined ephemeris, good check for accuracy)

Known Asteroid Relevant Functional Code:

```
def predict_improved(expnum, date, ra, dec):
    ra_1 = float(ra)
    dec_1 = float(dec)
    t = Time(date, format = 'mjd')
    jd = float(t.jd)
    p=propagate(np.array(known.a), np.array(known.e), np.array(known.i), np.array(known.w), np.array(known.W),
                np.array(known.M), np.array(known.epoch), np.zeros(len(known.a))+jd, helio=True)

    #mag = known.H + 5*np.log10(p.r*(p.delta))

    ra_matched = abs(p.ra - ra_1) < 0.0192
    dec_matched = abs(p.dec - dec_1) < 0.0192
    matched = ra_matched*dec_matched
    name_matches = []
    ra_matches = []
    dec_matches =[]
    expnum_matches = []
    date_matches = []
    epoch_matches = []
    exposure_ra = []
    exposure_dec = []
    if matched.sum() != 0:
        name_matches1 = list(known.name[matched])
        #name_matches_final = list(set(name_matches1)-set(name_matches))
        epoch_matches1 = list(known.epoch[matched])
        #epoch_matches_final = list(set(epoch_matches1)-set(epoch_matches))
        ra_matches1 = list(p.ra[matched])
        dec_matches1 = list(p.dec[matched])
        expnum_matches = [expnum]*len(name_matches)
        exposure_ra = [ra_1]*len(name_matches)
        exposure_dec = [dec_1]*len(name_matches)
        date_matches = [date]*len(name_matches)
    return  name_matches1, ra_matches, dec_matches, epoch_matches1
```

```
for ind, row in ephem.iterrows():
     if ind%10==0: print('Searching for observation #',ind, 'of MP ', MPname)
    pos_pred = SkyCoord(row['RA'], row['DEC'], unit=(u.deg, u.deg), frame='icrs')
    this_expnum = row['expnum']
    try:
         print(this_expnum, this_ccd, pos_pred.ra.deg, pos_pred.dec.deg, V_pred)
        dra = np.max([3, row['RA_3sigma']])/3600  # 1.0" min match
        ddec = np.max([3, row['DEC_3sigma']])/3600
        query = "select ra, dec, mag_auto, magerr_auto, expnum from UMTNO.DEEP_SE_OBJECT where \
        ra between "+str(pos_pred.ra.deg)+'-'+str(dra)+" and "+str(pos_pred.ra.deg)+'+'+str(dra) + " and
        dec between "+str(pos_pred.dec.deg)+'-'+str(ddec)+" and "+str(pos_pred.dec.deg)+'+'+str(ddec)+ \
        " and expnum = "+str(this_expnum)
        result = db.query_to_pandas(query)
        if (len(result)):
            matched +=1
#            print('matched! :)')
            #print(query)
            ra_obs.append(result.RA.values[0])
            dec_obs.append(result.DEC.values[0])
            mag_obs.append(result.MAG_AUTO.values[0])
            magerr_obs.append(result.MAGERR_AUTO.values[0])
        else:
#            print('not matched :(')
            ra_obs.append(np.nan)
            dec_obs.append(np.nan)
            mag_obs.append(np.nan)
            magerr_obs.append(np.nan)
    except:
        print('No data available for expnum ', this_expnum)
        pass
```

Ephemeride Generation, Known Asteroids:

2012 VX90

	expnum	targetnam	datetime_	RA	DEC	RA_3sigm	DEC_3sign	V
0	845639	353856 (20	2019-Apr-	215.857	-12.5765	0.147	0.128	21.69
0	845640	353856 (20	2019-Apr-	215.8567	-12.5764	0.147	0.128	21.69
0	845641	353856 (20	2019-Apr-	215.8565	-12.5762	0.147	0.128	21.69
0	845642	353856 (20	2019-Apr-	215.8562	-12.5761	0.147	0.128	21.69
0	845643	353856 (20	2019-Apr-	215.8559	-12.576	0.147	0.128	21.69
0	845644	353856 (20	2019-Apr-	215.8556	-12.5759	0.147	0.128	21.69
0	845645	353856 (20	2019-Apr-	215.8553	-12.5758	0.147	0.128	21.69
0	845646	353856 (20	2019-Apr-	215.855	-12.5757	0.147	0.128	21.69
0	845647	353856 (20	2019-Apr-	215.8548	-12.5756	0.147	0.128	21.69
0	845648	353856 (20	2019-Apr-	215.8545	-12.5754	0.147	0.128	21.69
0	845649	353856 (20	2019-Apr-	215.8542	-12.5753	0.147	0.128	21.69
0	845650	353856 (20	2019-Apr-	215.8539	-12.5752	0.147	0.128	21.69
0	845651	353856 (20	2019-Apr-	215.8536	-12.5751	0.147	0.128	21.69
0	845653	353856 (20	2019-Apr-	215.853	-12.5749	0.147	0.128	21.69
0	845652	353856 (20	2019-Apr-	215.8533	-12.575	0.147	0.128	21.69
0	845654	353856 (20	2019-Apr-	215.8528	-12.5748	0.147	0.128	21.69
0	845655	353856 (20	2019-Apr-	215.8525	-12.5746	0.147	0.128	21.69
0	845656	353856 (20	2019-Apr-	215.8522	-12.5745	0.147	0.128	21.69
0	845657	353856 (20	2019-Apr-	215.8519	-12.5744	0.147	0.128	21.69
0	845658	353856 (20	2019-Apr-	215.8516	-12.5743	0.147	0.128	21.69
0	845659	353856 (20	2019-Apr-	215.8514	-12.5742	0.147	0.128	21.69
0	845660	353856 (20	2019-Apr-	215.8511	-12.5741	0.147	0.128	21.69

2008 GP39

expnum	targetnam	datetime_	RA	DEC	RA_3sigm	DEC_3sign	V
845639	(2008 GP3	2019-Apr-	214.5658	-12.941	0.234	0.207	20.82
845640	(2008 GP3	2019-Apr-	214.5655	-12.9409	0.234	0.207	20.82
845641	(2008 GP3	2019-Apr-	214.5652	-12.9408	0.234	0.207	20.82
845642	(2008 GP3	2019-Apr-	214.565	-12.9408	0.234	0.207	20.82
845643	(2008 GP3	2019-Apr-	214.5647	-12.9407	0.234	0.207	20.82
845644	(2008 GP3	2019-Apr-	214.5645	-12.9406	0.234	0.207	20.82
845645	(2008 GP3	2019-Apr-	214.5642	-12.9406	0.234	0.207	20.82
845646	(2008 GP3	2019-Apr-	214.564	-12.9405	0.234	0.207	20.82
845647	(2008 GP3	2019-Apr-	214.5637	-12.9404	0.234	0.207	20.82
845648	(2008 GP3	2019-Apr-	214.5634	-12.9404	0.234	0.207	20.82
845649	(2008 GP3	2019-Apr-	214.5632	-12.9403	0.234	0.207	20.82
845650	(2008 GP3	2019-Apr-	214.5629	-12.9402	0.234	0.207	20.82
845651	(2008 GP3	2019-Apr-	214.5626	-12.9402	0.234	0.207	20.82
845653	(2008 GP3	2019-Apr-	214.5621	-12.94	0.234	0.207	20.82
845652	(2008 GP3	2019-Apr-	214.5624	-12.9401	0.234	0.207	20.82
845654	(2008 GP3	2019-Apr-	214.5619	-12.94	0.234	0.207	20.82
845655	(2008 GP3	2019-Apr-	214.5616	-12.9399	0.234	0.207	20.82
845656	(2008 GP3	2019-Apr-	214.5614	-12.9398	0.234	0.207	20.82
845657	(2008 GP3	2019-Apr-	214.5611	-12.9398	0.234	0.207	20.82
845658	(2008 GP3	2019-Apr-	214.5608	-12.9397	0.234	0.207	20.82
845659	(2008 GP3	2019-Apr-	214.5606	-12.9396	0.234	0.207	20.82
845660	(2008 GP3	2019-Apr-	214.5603	-12.9396	0.234	0.207	20.82

Relevant Code, Unknown Asteroids:

```
input_file1 = "DEEP_transients_A0b_20190402_CCD13.csv"
from skimage import io
transients = pd.read_csv(input_file1)
length = len(transients['EXPNUM'])

first_third = transients[0: int(length/3)]
second_third = transients[int(length/3) + 1: int(2*length/3)]
third_third = transients[int(2*length/3) + 1: int(length)]

fig, ax = plt.subplots(1, figsize=(20,10))
ax.plot(first_third['RA'], first_third['DEC'], '.r')
ax.plot(second_third['RA'], second_third['DEC'], '.b')
ax.plot(third_third['RA'], third_third['DEC'], '.g')

RA_numpy = transients['RA'].values
RA_numpy = np.round(RA_numpy, 4)
DEC_numpy = transients['DEC'].values
DEC_numpy = np.round(DEC_numpy, 4)
points = np.vstack((RA_numpy, DEC_numpy))
```

```
from PIL import Image
dec_size = int((np.amax(DEC_numpy) - np.amin(DEC_numpy))/0.0001)
ra_size = int((np.amax(RA_numpy) - np.amin(RA_numpy))/0.0001)
image = 0*(np.ndarray(shape = (dec_size,ra_size)))
```

```
print(np.shape(image))

for i in range(len(RA_numpy)):
    image_value1 = int((RA_numpy[i] - np.amin(RA_numpy))/0.0001)-1
    image_value2 = int((DEC_numpy[i] - np.amin(DEC_numpy))/0.0001)-1
    image[image_value2, image_value1] = 1
```

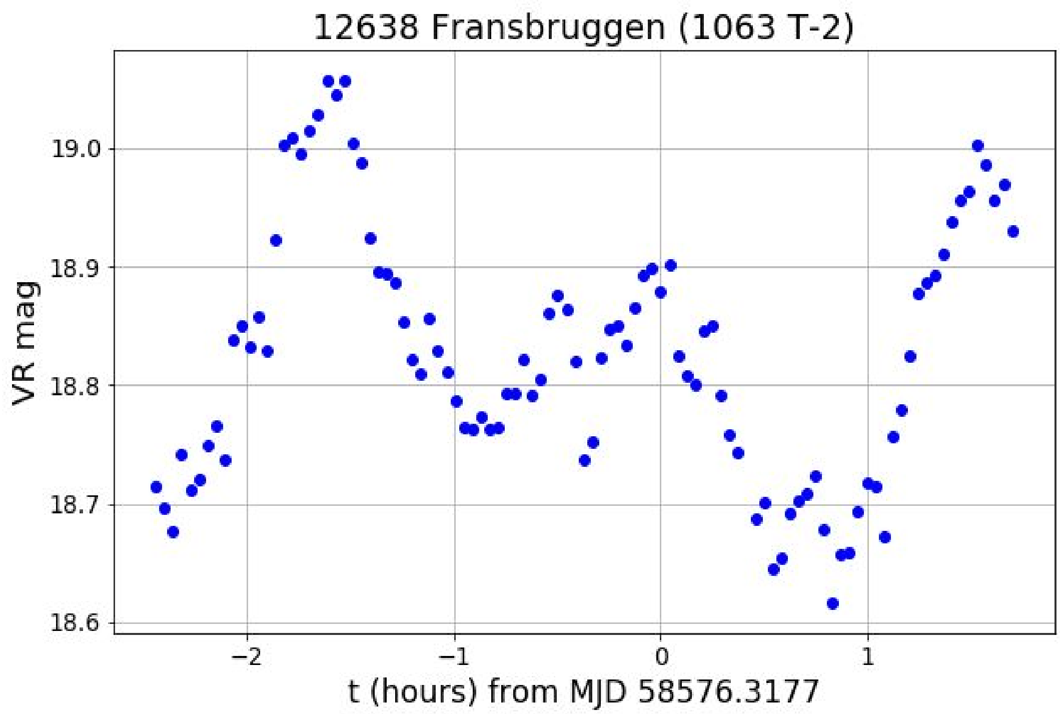
```
(1540, 3142)
```

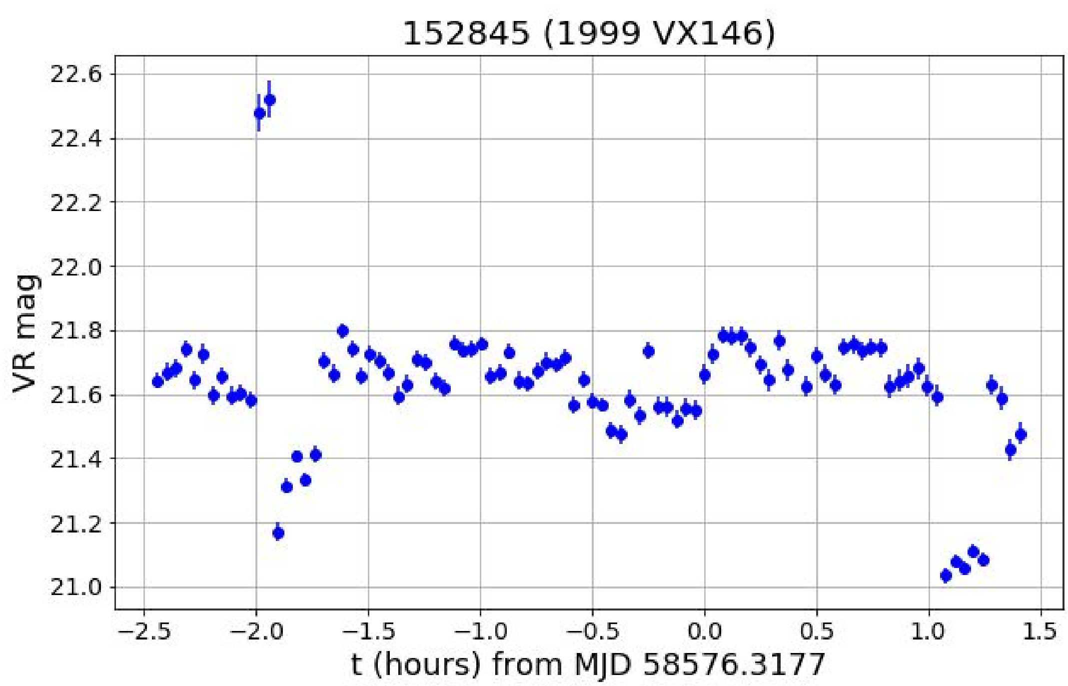
```
import matplotlib.cm as cm
image = image.astype(int)
#image = image[:, 0:370]
fig, ax = plt.subplots(figsize=(18, 10))
ax.imshow(image, origin = 'lower', cmap = cm.gray)
```

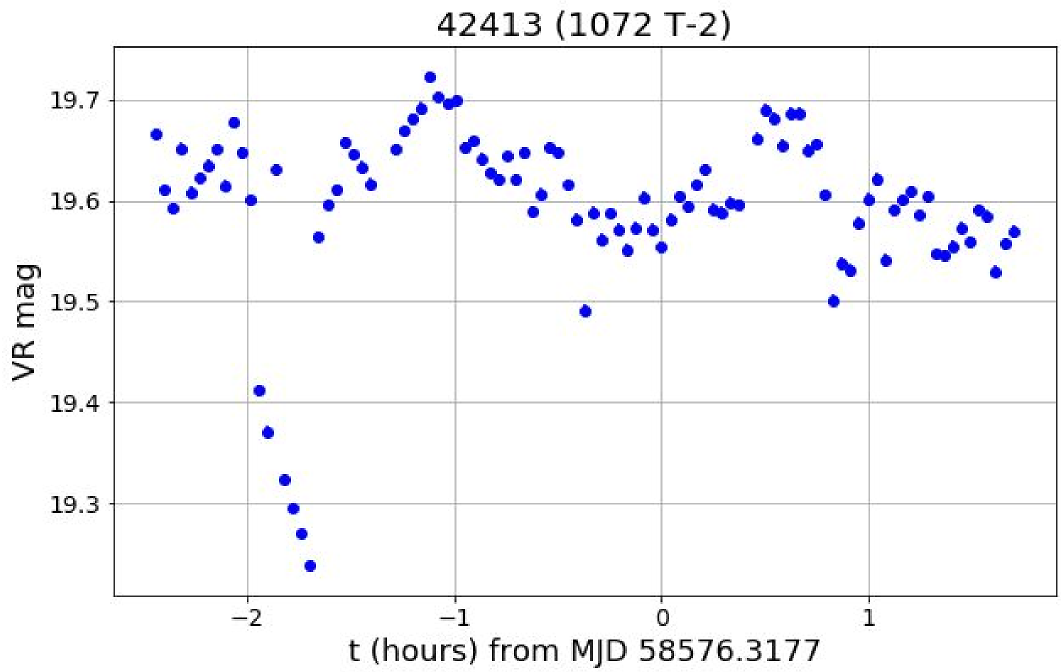
```python
from skimage.transform import probabilistic_hough_line, resize
from skimage.filters import median
from skimage.morphology import disk

# Line finding using the Probabilistic Hough Transform
#image = io.imread('CC13 Hough.JPG')
#image = image[:,:,1]
#image = median(image, disk(0.02))
from skimage.feature import canny

#edges = canny(image, 1, 1, 25)
lines = probabilistic_hough_line(image, threshold=5, line_length=3,
                                 line_gap=2)

# Generating figure 2
fig, ax = plt.subplots(1, 1, figsize=(18, 8), sharex=True, sharey=True)
import matplotlib.cm as cm
image1 = io.imread('CC13 Hough.jpg')
#image1 = resize(image1, (1540, 3142))
#ax.imshow(image1, cmap=cm.gray)
ax.set_title('Input image')


#ax[1].imshow(edges, cmap=cm.gray)
#ax[1].set_title('Canny edges')

#ax[2].imshow(edges * 0)
for line in lines:
    p0, p1 = line
    ax.plot((p0[0], p1[0]), (p0[1], p1[1]), color = 'red')
```

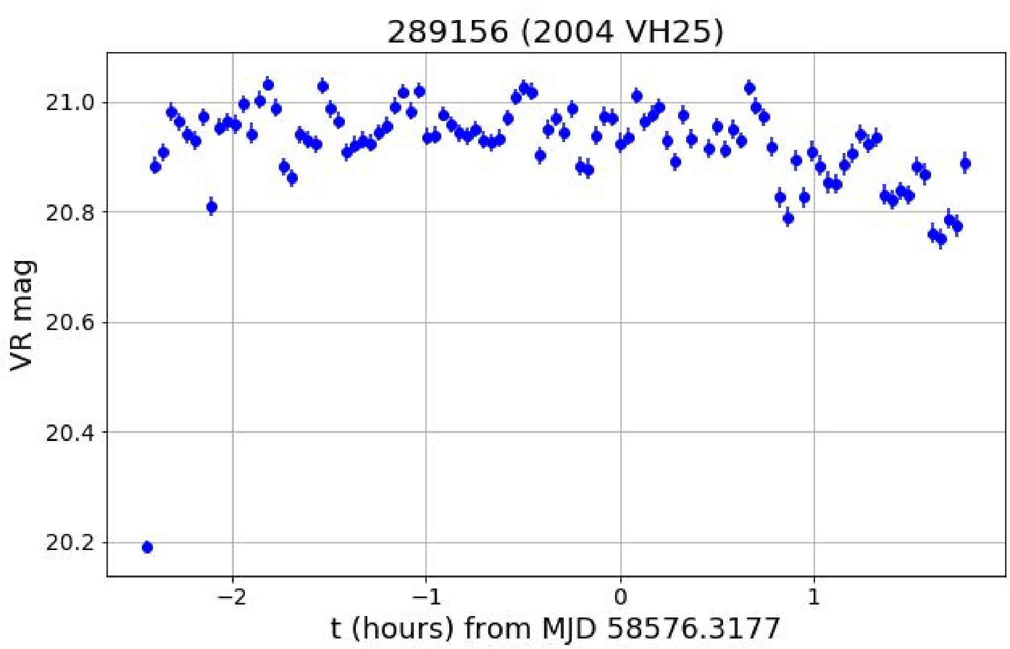
```python
l = refined_collections

out = []
while len(l)>0:
    first, *rest = l
    first = set(first)

    lf = -1
    while len(first)>lf:
        lf = len(first)

        rest2 = []
        for r in rest:
            if len(first.intersection(set(r)))>0:
                first |= set(r)
            else:
                rest2.append(r)
        rest = rest2

    out.append(first)
    l = rest
```

Generated Light Curves: *(4/390 Shown)*






42413 (1072 T-2)
VR mag
19.7
19.6
19.5
19.4
19.3
−2
−1
0
1
t (hours) from MJD 58576.3177


289156 (2004 VH25)
VR mag
21.0
20.8
20.6
20.4
20.2
−2
−1
0
1
t (hours) from MJD 58576.3177

__Appendix:__ More Code…

```
from propagate import propagate_lite as propagate

from propagate import propagate as propagate1

import pandas as pd

import numpy as np

from astropy.time import Time

from astropy.coordinates import SkyCoord

from astropy.time import Time

import time as TT

import rebound

from skyfield.api import Topos, Loader

from scipy.optimize import newton

from ccdBounds import *

import ephem

import csv

from itertools import compress

from astroquery.jplhorizons import Horizons

import astropy.units as u


exposures = pd.read_csv('DEEP_exposuretable_all.csv')

night = 20190402

field = 'A0b'



narrow_night = exposures.loc[exposures.loc[:, 'NITE']== night, :]

narrow_field = narrow_night.loc[narrow_night.loc[:, 'OBJECT']== field, :]

narrow_field


expnum = narrow_field['EXPNUM'].tolist()

date = narrow_field['MJD_OBS'].tolist()

ra = (narrow_field['RADEG']*(np.pi/180)).tolist()

dec = (narrow_field['DECDEG']*(np.pi/180)).tolist()


global known

known = pd.read_csv('MPCORB.csv', low_memory = False)


load = Loader('./Skyfield-Data', expire=False)

planets = load('de423.bsp')


name_to_number = {'0': 0, '1':1, '2':2, '3':3, '4':4, '5':5, '6':6, '7':7, '8':8, '9':9,'A':10,
        'B':11, 'C':12, 'D':13, 'E':14, 'F':15, 'G':16, 'H':17, 'I':18, 'J':19,
        'K':20, 'L':21, 'M':22, 'N':23, 'O':24, 'P':25, 'Q':26, 'R':27, 'S':28,
        'T':29, 'U':30, 'V':31,'W':32, 'X':33, 'Y':34, 'Z':35, 'a':36,
        'b':37, 'c':38, 'd':39, 'e':40, 'f':41, 'g':42, 'h':43, 'i':44, 'j':45,
        'k':46, 'l':47, 'm':48, 'n':49, 'o':50, 'p':51, 'q':52, 'r':53, 's':54,
        't':55, 'u':56, 'v':57, 'w': 58, 'x':59, 'y':60, 'z':61, }


def name_conversion(x):
```

```
        first = x[:1]
        first_number = int(name_to_number[first])*10000
        last = int(x[1:5])
        number = first_number + last
        return(number)


def compute_chip(rockra, rockdec, expra, expdec):
        '''
        Given the ra and dec of a point and of the center
        of an exposure, find the CCD containing that point.


        Returns a pair of the CCD name and number.
        '''
        deltara = 180/np.pi*ephem.degrees(rockra-expra).znorm
        deltadec = 180/np.pi*ephem.degrees(rockdec-expdec).znorm
        ccdname = 'None'
        for k in ccdBounds:
        if deltara > ccdBounds[k][0] and deltara < ccdBounds[k][1] and deltadec > ccdBounds[k][2] and deltadec < ccdBounds[k][3]:
        ccdname = k
        return ccdname, ccdNum[ccdname]


#global name_matches
#global epoch_matches
name_matches = []
ra_matches = []
dec_matches = []
expnum_matches  = []
date_matches = []
epoch_matches = []
exposure_ra = []
exposure_dec = []


def predict_improved(expnum, date, ra, dec):
        ra_1 = float(ra)
        dec_1 = float(dec)
        t = Time(date, format = 'mjd')
        jd = float(t.jd)
        p=propagate(np.array(known.a), np.array(known.e), np.array(known.i), np.array(known.w), np.array(known.W),
                np.array(known.M), np.array(known.epoch), np.zeros(len(known.a))+jd, helio=True)


        #mag = known.H + 5*np.log10(p.r*(p.delta))


        ra_matched = abs(p.ra - ra_1) < 0.0192
        dec_matched = abs(p.dec - dec_1) < 0.0192
        matched = ra_matched*dec_matched
        name_matches = []
        ra_matches = []
        dec_matches =[]
```

```
        expnum_matches = []
        date_matches = []
        epoch_matches = []
        exposure_ra = []
        exposure_dec = []
        if matched.sum() != 0:
        name_matches1 = list(known.name[matched])
        #name_matches_final = list(set(name_matches1)-set(name_matches))
        epoch_matches1 = list(known.epoch[matched])
        #epoch_matches_final = list(set(epoch_matches1)-set(epoch_matches))
        ra_matches1 = list(p.ra[matched])
        dec_matches1 = list(p.dec[matched])
        expnum_matches = [expnum]*len(name_matches)
        exposure_ra = [ra_1]*len(name_matches)
        exposure_dec = [dec_1]*len(name_matches)
        date_matches = [date]*len(name_matches)
        return  name_matches1, ra_matches, dec_matches, epoch_matches1

for n in range(len(expnum)):
        print(n)
        #print(name_matches)
        specific_exposure = expnum[n]
        specific_date = date[n]
        specific_ra = ra[n]
        specific_dec = dec[n]
        add_name, add_ra, add_dec,add_epoch, = predict_improved(specific_exposure, specific_date, specific_ra, specific_dec)
        add_expnum = [specific_exposure]*(abs(len(name_matches) -len(list(set(name_matches + add_name)))))
        add_expra = [specific_ra]*(abs(len(name_matches)-len(list(set(name_matches + add_name)))))
        add_expdec = [specific_dec]*(abs(len(name_matches)-len(list(set(name_matches + add_name)))))
        add_date = [specific_date]*(abs(len(name_matches)-len(list(set(name_matches + add_name)))))
        name_matches = list(set(name_matches + add_name))
        ra_matches = list(set(ra_matches + add_ra))
        dec_matches = list(set(dec_matches + add_dec))

expnum_matches.extend(add_expnum)
        date_matches.extend(add_date)
        epoch_matches = list(set(epoch_matches + add_epoch))
        exposure_ra.extend(add_expra)
        exposure_dec.extend(add_expdec)


def equa_to_ecl(X0,Y0,Z0):
        epsilon =  23.43929111 * np.pi/180.
        X = X0
        Y = Y0 * np.cos(epsilon) + Z0 * np.sin(epsilon)
```

```
    Z = -Y0 * np.sin(epsilon) + Z0 * np.cos(epsilon)
    return X, Y, Z


def xyz_to_kep(X, Y, Z, VX, VY, VZ, u):
    # compute the barycentric distance r
    r = (X**2 + Y**2 + Z**2)**0.5
    rrdot = (X*VX + Y*VY + Z*VZ)
    # compute the specific angular momentum h
    hx = Y * VZ - Z * VY
    hy = Z * VX - X * VZ
    hz = X * VY - Y * VX
    h = (hx**2 + hy**2 + hz**2)**0.5
    # compute eccentricity vector
    ex = (VY * hz - VZ * hy)/u - X/r
    ey = (VZ * hx - VX * hz)/u - Y/r
    ez = (VX * hy - VY * hx)/u - Z/r
    e = (ex**2+ey**2+ez**2)**0.5
    # compute vector n
    nx = -hy
    ny = hx
    nz = 0
    n = (nx**2 + ny**2)**0.5
    # compute true anomaly v, the angle between e and r
    v = np.arccos((ex * X + ey * Y + ez * Z) / (e*r))
    v[rrdot<0] = 2*np.pi - v[rrdot<0]
    # compute inclination
    i = np.arccos(hz/h)
    # compute eccentric anomaly E
    E = 2*np.arctan2((1-e)**0.5*np.sin(v/2.), (1+e)**0.5*np.cos(v/2.))
    # compute ascending node
    node = np.arccos(nx/n)
    node[ny<0] = 2*np.pi - node[ny<0]
    # compute argument of periapsis, the angle between e and n
    arg = np.arccos((nx * ex + ny * ey + nz *ez) / (n*e))
    arg[ez<0] = 2*np.pi - arg[ez<0]
    # compute mean anomaly
    M = E - e * np.sin(E)
    M[M<0] += 2*np.pi
    # compute a
    a = 1/(2/r - (VX**2+VY**2+VZ**2)/u)
    return a, e, i, arg, node, M


def rebound_simulation(epoch, date, input_x, input_y, input_z, input_vx, input_vy, input_vz):
    epoch0 = epoch #date of observation
    difference = float(epoch - date)
    #print(difference)
    ts = load.timescale()
    t = ts.tdb(jd=epoch0) #set epoch for simulation at t = epoch0
    Sun = planets['Sun']
    Mercury = planets[1]
    Venus = planets[2]
    Earth = planets[3]
    Mars = planets[4]
```

Jupiter = planets[5]

Saturn = planets[6]

Uranus = planets[7]

Neptune = planets[8]

sim = rebound.Simulation()

sim.units = ('day', 'AU', 'Msun')

sim.integrator = "IAS15"

Sun_x, Sun_y, Sun_z = Sun.at(t).position.au

Sun_vx, Sun_vy, Sun_vz = Sun.at(t).velocity.au_per_d

Sun_x, Sun_y, Sun_z = equa_to_ecl(Sun_x, Sun_y, Sun_z)

Sun_vx, Sun_vy, Sun_vz = equa_to_ecl(Sun_vx, Sun_vy, Sun_vz)

Sun_mass = 1

Mercury_x, Mercury_y, Mercury_z = Mercury.at(t).position.au

Mercury_vx, Mercury_vy, Mercury_vz = Mercury.at(t).velocity.au_per_d

Mercury_x, Mercury_y, Mercury_z = equa_to_ecl(Mercury_x, Mercury_y, Mercury_z)

Mercury_vx, Mercury_vy, Mercury_vz = equa_to_ecl(Mercury_vx, Mercury_vy, Mercury_vz)

Mercury_mass = 1.6601367952719304E-07

Venus_x, Venus_y, Venus_z = Venus.at(t).position.au

Venus_vx, Venus_vy, Venus_vz = Venus.at(t).velocity.au_per_d

Venus_x, Venus_y, Venus_z = equa_to_ecl(Venus_x, Venus_y, Venus_z)

Venus_vx, Venus_vy, Venus_vz = equa_to_ecl(Venus_vx, Venus_vy, Venus_vz)

Venus_mass = 2.4478383396645447E-06

Earth_x, Earth_y, Earth_z = Earth.at(t).position.au

Earth_vx, Earth_vy, Earth_vz = Earth.at(t).velocity.au_per_d

Earth_x, Earth_y, Earth_z = equa_to_ecl(Earth_x, Earth_y, Earth_z)

Earth_vx, Earth_vy, Earth_vz = equa_to_ecl(Earth_vx, Earth_vy, Earth_vz)

Earth_mass = 3.0404326462685257E-06

Mars_x, Mars_y, Mars_z = Mars.at(t).position.au

Mars_vx, Mars_vy, Mars_vz = Mars.at(t).velocity.au_per_d

Mars_x, Mars_y, Mars_z = equa_to_ecl(Mars_x, Mars_y, Mars_z)

Mars_vx, Mars_vy, Mars_vz = equa_to_ecl(Mars_vx, Mars_vy, Mars_vz)

Mars_mass = 3.2271514450538743E-07

Jupiter_x, Jupiter_y, Jupiter_z = Jupiter.at(t).position.au

Jupiter_vx, Jupiter_vy, Jupiter_vz = Jupiter.at(t).velocity.au_per_d

Jupiter_x, Jupiter_y, Jupiter_z = equa_to_ecl(Jupiter_x, Jupiter_y, Jupiter_z)

Jupiter_vx, Jupiter_vy, Jupiter_vz = equa_to_ecl(Jupiter_vx, Jupiter_vy, Jupiter_vz)

Jupiter_mass = 9.547919384243222E-04

Saturn_x, Saturn_y, Saturn_z = Saturn.at(t).position.au

Saturn_vx, Saturn_vy, Saturn_vz = Saturn.at(t).velocity.au_per_d

Saturn_x, Saturn_y, Saturn_z = equa_to_ecl(Saturn_x, Saturn_y, Saturn_z)

Saturn_vx, Saturn_vy, Saturn_vz = equa_to_ecl(Saturn_vx, Saturn_vy, Saturn_vz)

Saturn_mass = 2.858859806661029E-04

Uranus_x, Uranus_y, Uranus_z = Uranus.at(t).position.au

Uranus_vx, Uranus_vy, Uranus_vz = Uranus.at(t).velocity.au_per_d

Uranus_x, Uranus_y, Uranus_z = equa_to_ecl(Uranus_x, Uranus_y, Uranus_z)

Uranus_vx, Uranus_vy, Uranus_vz = equa_to_ecl(Uranus_vx, Uranus_vy, Uranus_vz)

Uranus_mass = 4.3662440433515637E-05

Neptune_x, Neptune_y, Neptune_z = Neptune.at(t).position.au

Neptune_vx, Neptune_vy, Neptune_vz = Neptune.at(t).velocity.au_per_d

Neptune_x, Neptune_y, Neptune_z = equa_to_ecl(Neptune_x, Neptune_y, Neptune_z)

Neptune_vx, Neptune_vy, Neptune_vz = equa_to_ecl(Neptune_vx, Neptune_vy, Neptune_vz)

Neptune_mass = 5.151389020535497E-05

sim.add(m=Sun_mass, x=Sun_x, y=Sun_y, z=Sun_z, vx=Sun_vx, vy=Sun_vy, vz=Sun_vz)

sim.add(m=Mercury_mass, x=Mercury_x, y=Mercury_y, z=Mercury_z, vx=Mercury_vx, vy=Mercury_vy, vz=Mercury_vz)

sim.add(m=Venus_mass, x=Venus_x, y=Venus_y, z=Venus_z, vx=Venus_vx, vy=Venus_vy, vz=Venus_vz)

sim.add(m=Earth_mass, x=Earth_x, y=Earth_y, z=Earth_z, vx=Earth_vx, vy=Earth_vy, vz=Earth_vz)

sim.add(m=Mars_mass, x=Mars_x, y=Mars_y, z=Mars_z, vx=Mars_vx, vy=Mars_vy, vz=Mars_vz)

sim.add(m=Jupiter_mass, x=Jupiter_x, y=Jupiter_y, z=Jupiter_z, vx=Jupiter_vx, vy=Jupiter_vy, vz=Jupiter_vz)

sim.add(m=Saturn_mass, x=Saturn_x, y=Saturn_y, z=Saturn_z, vx=Saturn_vx, vy=Saturn_vy, vz=Saturn_vz)

sim.add(m=Uranus_mass, x=Uranus_x, y=Uranus_y, z=Uranus_z, vx=Uranus_vx, vy=Uranus_vy, vz=Uranus_vz)

sim.add(m=Neptune_mass, x=Neptune_x, y=Neptune_y, z=Neptune_z,

```
vx=Neptune_vx, vy=Neptune_vy,
vz=Neptune_vz)


        X, Y, Z = input_x, input_y, input_z
        VX, VY, VZ = input_vx, input_vy,
input_vz
        sim.add(m=0, x=X, y=Y, z=Z,
vx=VX, vy=VY, vz=VZ)

        sim.integrate(-difference,
exact_finish_time=1)

        asteroid = sim.particles[-1]

        u_bary = 2.9630927492415936E-04 #
standard gravitational parameter, GM. M is
the mass of sun + all planets
        x1, y1, z1 = np.array([asteroid.x]),
np.array([asteroid.y]), np.array([asteroid.z])
        vx1, vy1, vz1 =
np.array([asteroid.vx]),
np.array([asteroid.vy]),
np.array([asteroid.vz])
        a, e, i, arg, node, M = xyz_to_kep(x1,
y1, z1, vx1, vy1, vz1, u_bary)

        b = propagate1(a, e, i, arg, node, M,
epoch-difference, epoch-difference, helio =
False)
        ra = float(b.ra)
        dec = float(b.dec)
        return ra, dec

object_name = []
ra_matched1 = []
dec_matched1 = []
for i in range(len(name_matches)):
        object_name = [name_matches[i]]
        asteroids =
known[known.name.isin(object_name)]
        store = float(asteroids.epoch)
        w = propagate1(np.array(asteroids.a),
np.array(asteroids.e), np.array(asteroids.i),
np.array(asteroids.w), np.array(asteroids.W),
np.array(asteroids.M),
np.array(asteroids.epoch),
np.array(asteroids.epoch), helio=True)
        input_x, input_y, input_z, input_vx,
input_vy, input_vz = w.X, w.Y, w.Z, w.VX,
w.VY, w.VZ

        t1 = Time(date_matches[i], format =
'mjd')
        jd1 = float(t1.jd)
        #print(jd1)

        ra_match, dec_match =
rebound_simulation(store, jd1, input_x,
input_y, input_z, input_vx, input_vy,
input_vz)
        ra_matched1.append(ra_match)
        dec_matched1.append(dec_match)

chip_name_matches = []
chip_number_matches = []
check_ra = 0
check_dec = 0
check_expra = 0
check_expdec = 0
```

```
for i in range(len(name_matches)):
    check_ra = ra_matched1[i]
    check_dec = dec_matched1[i]
    check_expra = exposure_ra[i]
    check_expdec = exposure_dec[i]
    chipname, chipnumber =
compute_chip(check_ra, check_dec,
check_expra, check_expdec)

chip_name_matches.append(chipname)

chip_number_matches.append(chipnumber)


ra_matched_deg = [i * 180/np.pi for i in
ra_matched1]
dec_matched_deg =[i * 180/np.pi for i in
dec_matched1]
exposure_ra_deg = [i * 180/np.pi for i in
exposure_ra]
exposure_dec_deg = [i * 180/np.pi for i in
exposure_dec]


for i in range(len(name_matches)):
    match_name = name_matches[i]
    match_name = match_name.rstrip()
    name_matches[i] = match_name



ra_3sigma = []
dec_3sigma = []
ra_horizons = []
dec_horizons = []
target_name = []


for i in range(len(name_matches)):
    print(i)
    match_name = name_matches[i]
    if len(match_name) <= 5:
    match_name =
str(name_conversion(match_name))
    date1 = date_matches[i]
    intermediate = Time(date1, format =
'mjd')
    intermediate_jd =
Time(intermediate.jd + 0.5, format = 'jd')
    date2 = intermediate_jd.iso
    obj =
Horizons(id=match_name,epochs={'start':int
ermediate.iso, 'stop':date2,'step':'3h'})
    try:
    eph = obj.ephemerides()
    ra3sigmalist = eph['RA_3sigma']
    dec3sigmalist = eph['DEC_3sigma']
    rahorizonslist = eph['RA']
    dechorizonslist = eph['DEC']
    targetnamelist = eph['targetname']
    ra3sigma = ra3sigmalist[0]
    dec3sigma = dec3sigmalist[0]
    rahorizons = rahorizonslist[0]
    dechorizons = dechorizonslist[0]
    targetname = targetnamelist[0]
    ra_3sigma.append(ra3sigma)
    dec_3sigma.append(dec3sigma)
    ra_horizons.append(rahorizons)
    dec_horizons.append(dechorizons)
```

```
        target_name.append(targetname)
    except ValueError:
        ra3sigma = 'Unknown'
        dec3sigma = 'Unknown'
        rahorizons = 'Unknown'
        dechorizons = 'Unknown'
        targetname = 'Unknown'
        ra_3sigma.append(ra3sigma)
        dec_3sigma.append(dec3sigma)
        ra_horizons.append(rahorizons)
        dec_horizons.append(dechorizons)
        target_name.append(targetname)

name_matches.insert(0, ' MPC Object Name')
target_name.insert(0, 'Principal Designation')
expnum_matches.insert(0, 'Exposure Number')
date_matches.insert(0, 'Exposure Date')
ra_matched_deg.insert(0, 'RA of Object (deg)')
ra_3sigma.insert(0, '3 Sigma Uncertainty of RA (arcsec)')
dec_matched_deg.insert(0, 'DEC of Object (deg)')
dec_3sigma.insert(0, '3 Sigma Uncertainty of DEC (arcsec)')
exposure_ra_deg.insert(0, 'Exposure RA Center (deg)')
exposure_dec_deg.insert(0, 'Exposure DEC Center (deg)')
chip_name_matches.insert(0, 'DES Chip Name of Matched Object')
chip_number_matches.insert(0, 'DES Chip Number of Matched Object')
ra_horizons.insert(0, 'RA Horizons (Deg)')
dec_horizons.insert(0, 'DEC Horizons (Deg)')

df4 = pd.DataFrame(list(zip(*[name_matches, target_name, expnum_matches, date_matches, ra_matched_deg, ra_horizons, ra_3sigma, dec_matched_deg, dec_horizons, dec_3sigma, exposure_ra_deg, exposure_dec_deg, chip_name_matches, chip_number_matches])))
```

SECTION 2: Generate Ephemerides, Make Light Curves!

```
from propagate import propagate_lite as propagate
from propagate import propagate as propagate1
import pandas as pd
import numpy as np
from astropy.time import Time
from astropy.coordinates import SkyCoord
from astropy.time import Time
import time as TT
import rebound
from skyfield.api import Topos, Loader
from scipy.optimize import newton
from ccdBounds import *
import ephem
import csv
from itertools import compress
from astroquery.jplhorizons import Horizons
import easyaccess as ea
```

```
import datetime as dt
import astropy.units as u
import matplotlib.pyplot as plt
import matplotlib.patches as patches
import cv2 as cv


night = 20190402
field = 'A0b'
filename = 'Matches DEEP, ' + str(field) + ', ' + str(night) + '.csv'
matches = open(filename)
csv_f = csv.reader(matches)
names = []


for row in csv_f:
	names.append(row[2])


db = ea.connect(section = 'umtno')
del(names[0])
del(names[0])
length = len(names)


for i in range(length):
	print(i)
	ra_obs = []
	dec_obs = []
	mag_obs = []
	magerr_obs = []
	matched = 0
	name = str(names[i])
	#print(name)
	# Convert time to iso format from non-standard Horizons format:
	input_file = str(field) + " " + str(night)+ " "  + str(name) + " " + "Ephemerides" + ".csv"
	ephem = pd.read_csv(input_file)


	dates = [dt.datetime.strptime(d, '%Y-%b-%d %H:%M:%S.%f') for d in ephem['datetime_str']]
	dates = Time([d.strftime('%Y-%m-%d %H:%M:%S.%f') for d in dates], format='iso', scale='utc')
	mjd = dates.mjd
	MPname = ephem['targetname'][0]
	V_pred = ephem['V'][0]
	print(V_pred)


	for ind, row in ephem.iterrows():
#		if ind%10==0: print('Searching for observation #',ind, 'of MP ', MPname)
	pos_pred = SkyCoord(row['RA'], row['DEC'], unit=(u.deg, u.deg), frame='icrs')
	this_expnum = row['expnum']
	try:
#		print(this_expnum, this_ccd, pos_pred.ra.deg, pos_pred.dec.deg, V_pred)
	dra = np.max([3, row['RA_3sigma']])/3600  # 1.0" min match
	ddec = np.max([3, row['DEC_3sigma']])/3600
	query = "select ra, dec, mag_auto, magerr_auto, expnum from UMTNO.DEEP_SE_OBJECT where \
```

```
ra between
"+str(pos_pred.ra.deg)+'-'+str(dra)+" and
"+str(pos_pred.ra.deg)+'+'+str(dra) + " and  \
        dec between
"+str(pos_pred.dec.deg)+'-'+str(ddec)+" and
"+str(pos_pred.dec.deg)+'+'+str(ddec)+ \
        " and expnum = "+str(this_expnum)
        result = db.query_to_pandas(query)
        if (len(result)):
                matched +=1
#               print('matched! :)')
                #print(query)

ra_obs.append(result.RA.values[0])

dec_obs.append(result.DEC.values[0])

mag_obs.append(result.MAG_AUTO.values[
0])

magerr_obs.append(result.MAGERR_AUTO
.values[0])
        else:
#               print('not matched :(')
                ra_obs.append(np.nan)
                dec_obs.append(np.nan)
                mag_obs.append(np.nan)
                magerr_obs.append(np.nan)
        except:
        print('No data available for expnum ',
this_expnum)
        pass
        print('Matched ', matched,
'observations for', MPname)
        if matched>0:
        print(mag_obs)
        fig, ax = plt.subplots(1,
figsize=(10,6))
        good_mag =
np.where(np.abs(mag_obs-V_pred)<2)

ax.errorbar((np.array(mjd)[good_mag]-mjd[0
])*24,
np.array(mag_obs)[good_mag],yerr=np.array
(magerr_obs)[good_mag], fmt='o', color='b')
        ax.tick_params(axis='both',
which='major', labelsize=14)
        ax.tick_params(axis='both',
which='minor', labelsize=14)
        ax.set_title(MPname, fontsize=20)
        ax.grid()
        ax.set_xlabel('t (hours) from MJD
'+str(round(mjd[0],4)), fontsize=18)
        ax.set_ylabel('VR mag', fontsize=18)
        fout = str(field) + " " + str(night)+ "1
"  + str(name) + " " +  "Light Curve"
        plt.savefig(fout+'.png')
        plt.clf()
```