Hindsight

Deck 36, Main Engineering

Revenge of the space book!

Due to budget cutbacks, the optical sensor on your satellite was made with parts from the flea market. This sensor is terrible at finding stars in the field of view. Readings of individual positions of stars is very noisy. To make matters worse, the sensor can't even measure magnitudes of stars either.

Budget cutbacks aside, make your shareholders proud by identifying which stars the sensor is looking at based on the provided star catalog.

General Information

Hindsight was a challenge about finding the attitude of a satellite, quite similar to a series of similar challenges during last years CTF. As a little help one was given a star catalog of unit vectors, however as a bit of added difficulty (and as challenge flavor) the magnitudes of the stars were ommitted.

Once you connected to the challenge you were given a further set of unit vectors. These were a subset of the same star-catalog, however these were (obviously) from the satellites perspective, and as such rotated by the satellites attitude.

Theory

For matching stars purely upon their unit vectors, it is helpful to find a relation that is perserved upon a rotation of all inputs. Probably the easiest relation that fulfills that criterion is the angle between two vectors (With R being some rotation matrix):

$$cos(\alpha) = \frac{v_1 v_2}{||v_1|| ||v_2||} = \frac{(Rv_1)(Rv_2)}{||Rv_1|| ||Rv_2||}$$

Thus the angles between the stars given by the challenge S should be a subset of the angles between all stars in the catalog C. Since we are given unit-vectors we use the scalar product of these vectors as a direct proxy for the angle.

If we assume the catalog to be a $N \times 3$ matrix our angle-proxy (CA) can be calculated as:

$$A_C = CC^T$$

We can calculate the same metric on the stars given by the challenge.

$$A_S = SS^T$$

If we now want to find the angle between Star A and Star B we look into the Ath column, row B, to find the desired angle. As stated above the Matrix A_S should be a subset of A_C, or in other words there has to be a Indexing-Vector I, such that (numpy-Syntax):

$$A_C[I,I] = A_S$$

Once I is found we know that the nth star in S corresponds to the I[n]-th star in C. The general rotation can now be found by using the Kabsch-algorithm between C[I] and S.

Finding I

I used a greedy, recursive matching algorithm for finding the indexing-vector. Let's first assume that we've found a match between the star S[0] and $C[x_0]$. We can now apply a constraint upon all further stars that the angle between S[1] and S[0] must be the approximately the same between $C[x_1]$ and $C[x_0]$. For all stars x_i in the catalog that fulfill this condition, we go into the next recursion-level and try to again to find all new stars that match the condition for all previously matched stars. In most cases no match is found in the end, in which case some previously determined star is probably wrong, and so we just return and try the next one. If all stars have been matched, we've succeeded and return the guessed stars $[x_0, ..., x_N]$.

```
def test():
    for start_idx in range(CATALOG_LEN):
        guess = [ start_idx ]
        res = test_val(guess)
        if res != []:
            print(res)
            return res
def test_val(guesses):
    # print(len(quesses), quesses)
    solutions = []
    for test_idx in range(CATALOG_LEN):
        if test_idx in guesses:
            continue
        if sum([abs( catalog_angles[guess, test_idx] - testval_angles[ len(guesses), guess_idx ] ) for gue
            next_guesses = guesses + [ test_idx ]
            if len(next_guesses) >= testval_angles.shape[1]:
                return [ next_guesses ]
            #print(next_quesses)
            solutions += test_val(next_guesses)
    return solutions
Thus we know I and can calculate the Satellite rotation.
Implementation
import numpy as np
from telnetlib import Telnet
with open("catalog-0eafe1f09a48df8cf6299fa88c2262df2b004c30.txt") as f:
    txt = [ [float(val) for val in i.split(",\t") ] for i in filter(lambda a: a.strip() != "", f.readlin
# testvals = [
     [ 0.063510,
                                      0.983013 ],
                       0.172198,
     [ 0.099288,
                                      0.979356],
#
                       -0.176080,
#
     [-0.055440,
                       -0.154325,
                                      0.986464],
#
     [ 0.125521,
                       -0.002894,
                                      0.992087],
     [ -0.218239,
                                      0.973591 ],
                       -0.067029,
#
     [ -0.018649,
                       0.063052,
                                      0.997836],
#
     [ -0.074830,
                       0.156673,
                                      0.984812 ],
```

0.996748],

0.980626],

0.980111],

0.991518],

0.985048],

0.995450],

0.967076],

0.984494],

0.984511],

0.984511], 0.973503], 0.999743],

0.981127],

0.966652],

[-0.035332,

[0.083429,

[0.123108,

[-0.172270,

[0.095199,

[-0.061920,

[-0.165385,

[-0.158970,

[-0.164970,

[-0.046028,

[0.020075,

[-0.030429,

[0.026359,

0.072418,

-0.177233,

-0.155649,

0.127270,

-0.002115,

0.003956,

-0.058480,

-0.110085,

0.059348,

-0.223992,

-0.010497,

-0.254281,

0.246842,

#

#

#

#

#

#

#

#

#

#

#

#

```
-0.125803, 0.991431 ],

-0.053336, 0.993071 ],

0.069087, 0.987212 ],

-0.233888, 0.972243 ],

-0.123531, 0.984928 ],

-0.009800, 0.971845 ],
     [ 0.035101,
[ 0.104716,
#
#
#
     [ -0.006348,
#
#
      [ -0.121062,
#
     [ 0.235419,
     [ -0.130585,
                                          0.991385],
#
                         -0.010126,
      [ 0.111752, -0.184471, 0.976464 ],
[ 0.064663, 0.160371, 0.984936 ],
[ 0.108471, 0.068876, 0.991711 ],
     [ 0.111752,
#
#
     [ 0.064663,
#
# ]
catalog = np.array( txt )
catalog = np.divide( np.array( catalog ), np.sum(catalog**2, axis = 1, keepdims=True) )
print(catalog.shape)
# testvals = np.array( testvals )
# testvals = np.divide( np.array( testvals ), np.sum(testvals**2, axis = 1, keepdims=True) )
# print(testvals.shape)
catalog_angles = np.matmul(catalog, catalog.transpose())
# testval_angles = np.matmul(testvals, testvals.transpose())
CATALOG_LEN = catalog_angles.shape[0]
ERROR\_THRESH = 0.0005
def test_val(guesses):
    # print(len(guesses), guesses)
    solutions = []
    for test_idx in range(CATALOG_LEN):
         if test_idx in guesses:
             continue
         if sum([abs( catalog_angles[guess, test_idx] - testval_angles[ len(guesses), guess_idx ] ) for gue
             next_guesses = guesses + [ test_idx ]
             if len(next_guesses) >= testval_angles.shape[1]:
                  return [ next_guesses ]
              #print(next_quesses)
             solutions += test_val(next_guesses)
    return solutions
# for start_idx in range(CATALOG_LEN):
     guess = [ start_idx ]
#
     res = test\_val(guess)
     if res != []:
          print(res)
def test():
    for start_idx in range(CATALOG_LEN):
         guess = [ start_idx ]
         res = test_val(guess)
```

```
if res != []:
            print(res)
            return res
# 25, 51, 180, 235, 277, 311, 318, 479, 538, 693, 715, 775, 852, 944, 979, 992, 1128, 1134, 1151, 1218, 12
# 12, 33, 53, 130, 136, 140, 213, 226, 458, 563, 575, 593, 627, 649, 659, 682, 699, 703, 705, 874, 1014, 1
with Telnet( "early-motive.satellitesabove.me", 5002) as tn:
    tn.read_until(b"please:\n")
    tn.write(b"ticket{whiskey65795tango2:GD69RsTs54vMK5B14BL6CT8c837Q1XDb6_NJcEAI3zI2ZeMfEf_x4A_z4_EEo3YRa
    for i in range(5):
       print(i)
        testvals = []
        while True:
            line = tn.read_until(b"\n").strip().decode()
            if line == "":
            testvals.append([ float(val.strip()) for val in line.split(",") ])
        testvals = np.array( testvals )
        testvals = np.divide( np.array( testvals ), np.sum(testvals**2, axis = 1, keepdims=True) )
        testvals = np.array( testvals )
        testvals = np.divide( np.array( testvals ), np.sum(testvals**2, axis = 1, keepdims=True) )
        testval_angles = np.matmul(testvals, testvals.transpose())
        tn.read until(b"):\n")
        res = test()
        if res is None:
            print("MATCH FAILED for ", testvals)
           raise Exception()
        tn.write(', '.join([ str(val) for val in res[0]] ).encode() + b"\n")
        print(tn.read_until(b"\n"))
    print(tn.read_until(b"\n"))
    while True:
        buf = b""
        bytes_read = tn.read_some()
        buf += bytes_read
        if bytes_read == b"":
            break
    print(buf.decode())
```