Star Mapping Algorithm

Darick W. LaSelle
dlaselle@stanford.edu (alt dwlaselle@gmail.com)

This paper proposes a project investigating how to map a set of points (presumed to be stars) onto a star map. I will implement an algorithm that does star mapping against an existing database, using a point matching algorithm.

Goal 1:

Develop a database that can accurately represent the stars both numerically and visually. This will include identifying a way to visually display brightness. The database assembled comes from the Saguaro Astronomy club [3]. I included a script that loads the data in from a "manually pre-filtered" csv file. This script will cover the numeric representation of the stars. In addition, this script is a precursor to "fingerprint files". That is, it is a representation of the position of all of the stars within a specified area of the sky (entered in radians). Over the course of the project, I will run this algorithm multiple times to assess different sizes of star maps (i.e. minimalist to test functionality, a complete map, the visual sky, etc. This will also provide the ability to produce a star map around each star, should it be desired (this will be useful in truth-testing.

Goal 2:

Be able to correctly map a "mock up" test map and match that to the database.

Goal 3:

Be able to correctly map a distorted (i.e. skewed images, added noise, etc) to the database.

Goal 4:

Be able to map real pictures of stars and match the database.

Goals 2 through 4 will be accomplished using a point matching algorithm. Point matching algorithms have existed, and been well vetted, for many years. As a specific example, the Institute for Aerospace and Astronautics of theTechnical University of Berlin uses point matching to measure position and orientation on their satellites. [1] Additional work on this application was published in 1994 in Academic Press Limited. [2] This early work identifies that the concept is feasible, even with limited computing power. However, I was not able to locate any detailed descriptions of the algorithms. The majority of the code developed for the star matching system and the data used to test the system has been organically developed.

The use of point matching to identify patterns will roughly follow the following algorithm:

- Identify a point
- Find an adjacent point (neighbor), and set it as a reference neighbor
- Find additional neighbors
- Match the parameters to associated truth files
- Score the comparisons
- Select a best match (if one exists)

Potential Roadblocks

- Image angles – while this point matching algorithm is intended to be independent of picture orientation, that will need to be vetted
- Obstructions – If a star is covered by a tree or a cloud, a point in the fingerprint file will be unaccounted for. This is also true for low intensity stars
- Noise – Reflections, satellites, planes, etc can also cause additional points to be found
- Processing power – This algorithm may shape up to be an σ^4 operation on a large database (at a minimum there are assumed to be 500 visible stars, while the database has 28,000+stars). Short of moving to a parallel processing program, the run time on this algorithms are likely to be very time and processor intensive

Additional Notes

- Once the algorithm is running correctly for the point matching, a number of image processing techniques can be used in order to generate a "map" of the stars from a real image. Notably, corner and feature detection could be used here. However, at the time of this proposal, I believe that grayscale thresholding would be the first step, followed by a series of opening and closing filters. At that point, either through further erosion, or region labelling, I would reduce this to a point map (the Matlab "imregionalmax" command could be useful here)
- A point matching algorithm has been successfully used in star finding algorithms. The "TUBSAT" project, which was published in 1994 used this concept (http://static.aminer.org/pdf/PDF/000/994/725/an_application_of_point_pattern_matching_in_astronautics.pdf)
- The idea of using a point matching technique is to be more robust and more specific than using feature based image matching based on constellations. In particular, if only a partial constellation is visible, or there are multiple constellations, feature matching may fail. However, point matching should theoretically be more resistant to this, as it will be based on stars, and can by definition is only going to test a star that it can see.

  Note: If I am unable to progress relatively quickly through this concept, I will switch to using a constellation database.

[1] V. Paquin, "*Point Pattern Matching*," [Online]. Available: http://www3.sympatico.ca/vpaquin/tutorial/

[2] G. Weber, L. Knipping, H. Alt, "*An application of point pattern matching in astonautics"* Aerospace and Astronautics of theTechnical University of Berlin, J. Symbolic Computation (1994) 11

[3] S. Coe, "*Saguaro Astronomy Club Database version 8.1*," [Online]. Available: http://www.saguaroastro.org/content/downloads.htm

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Final Project              %
%                            %
% "The Star Finder"          %
%                            %
% An Algorithm to Identify   %
% Stars From a Digital Image %
%                            %
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clear all
clc
close all
tic;
% Read the csv file formatted file should have four data columns:
% Azimuth (hours), Azimuth (minutes), Altidude(degrees), Altitude
(minutes)
% All in standard J2000 Celestial Coordinate formatting
% The fifth column inidicated weather the declination is positive
(+1) or
% negative (-1)
data_in=csvread('StarDataIn.csv');

%convert input file into spherical coordinates
data_out = [360*data_in(:,1)/24+360*data_in(:,2)/(24*60),
(data_in(:,3)+data_in(:,4)/(60)).*data_in(:,5)];

% number of degrees that the point matched with have to be within
deg_rad = 3;

% Find out how many stars there are
num_stars=length(data_in);

for i=1:num_stars %change to num_stars later
    k=3;
    for j=1:num_stars
        if (i~=j)

dist_test=[data_out(i,1),data_out(i,2);data_out(j,1),data_out(j,2)]
;
            if (pdist(dist_test)<deg_rad)
                x_dist = dist_test(2,1)-dist_test(1,1);
                y_dist = dist_test(2,2)-dist_test(1,2);
                [theta,rho] = cart2pol(x_dist,y_dist);
                data_out(i,k) = 180*theta/pi;
                data_out(i,k+1) = rho;
                k = k+2;
            end
        end
    end
end
csvwrite('StarDataOut_test.csv',data_out);
time=toc
```

This first run took 152 second to run. It produced a file 168 fields wide, which simply means that one star had 83 neighbors.