**How we tackled the Lost-in-Space algorithm**

**Breakdown:**

1. Image processing and Feature extraction
2. Finding the Angular distances
3. Mapping image stars to their IDs
4. Calculating the attitude

**Image processing and Feature extraction**

Ivo write your stuff here

**Finding the Angular distances**

Since our camera has a diagonal FOV of about 78.6 degrees the images it’ll produce will be almost identical to the gnomonic projection of the stars in our FOV. This allows us to assume a pinhole model of a camera to find the angular distances between stars in our image without accounting for warping.

It allows us to assume a single point through which the light enters our sensor. After which almost all of the needed camera parameters can be calculated using trigonometry ( This is useful because our only way of testing whether or not our approach is correct was through Stellarium which doesn’t give a lot of information except for the vertical FOV and the resolution of the image)

We compute the unit vector for each star selected in the step above using it’s x, y pixel coordinates and the focal length by x or y (since our sensor is not square) as the coordinate of the z axis. Then we normalize the vectors

To find the angular distances between each pair of stars after that becomes a problem of just computing the arcscosine of the dot product of both unit vectors.

The angular distance between two stars in our image is very valuable since it becomes the main reliable parameter by which to determine the stars in our image

**Mapping image stars to their IDs**

Now that we have the angular distance for each pair of stars in our image in a hypothetical perfect scenario we can immediately know which pair of stars in our database corresponds to the one in our image. However we run into a few problems:

1. For each pair of stars chosen from our image we have the angular distance calculated, however due to lens distortion and various other factors the offset of some of those stars’s angular distances can reach close to 2 degrees
2. Even if for each pair of stars in our image you had exactly one candidate pair from the database you still wouldn’t know which star in the image pair corresponds to which ID in the database pair
3. In practice due to the angular distance offset the candidates for each image pair become a lot more than one or two since we are working with some tolerances

* **How do we solve these problems to get the final mapping?**

We decided to look at the problem through the lens of graph theory. If we represent each image star *si* as a node in a graph with it’s adjacency list being all the possible ID candidates for this star the problem becomes that of graph traversal. We need to start going through the graph, taking a star and one of it’s IDs adding it to the current possible assignment and checking if it meets the geometric requirements of the rest of the assignment so far. For now we don’t need to think about one perfect and full assignment (by full I mean every image star having a catalog ID assignment), we just need to keep track of all correct assignments so far that meet our requirements (Note that we need at least three identified stars to determine our orientation). In our most current implementation we use a DFS (Depth First Search) algorithm with a minimum remaining value heuristic and a hashmap to represent our graph.

Now that we have all possible incomplete or complete assignments we have to figure out how to determine which is the best one. To do that we implemented a scoring system that gives a grade to each solution based on two main factors: difference between image and catalog angular distance and amount of stars identified. A perfect solution that satisfies all requirements would have a score of 0, so in hindsight he smaller the score the better the solution. Which is why out of our solutions we choose the one with the lowest grade.

Now that we have a solution of at least three mappings there’s the problem of outlier assignments that are incorrect. That’s the reason why in practice we make the minimum assignments for each solution a bit higher than 3. We’ll look at a way of tackling the problem outliers in the next section

**Calculating the attitude**

For our application we decided to use the QUEST algorithm to get the final orientation in quaternion form from the unit vectors in our image and the unit vectors of assigned star IDs we get from our database. We decided to calculate the attitude in quaternion form since our IMU already takes quaternions to be calibrated. During the QUEST algorithm we build the weighted attitude profile matrix B which takes into account each vector pairs weight (We effectively ignore a pair if it’s weight is 0). After which we construct Davenport’s K matrix and sove for it’s maximum eigenvalue essentially giving us our quaternion. In our most recent version we don’t have different weights for each pair, trusting them all equally and giving them a weight of 1, but we are oriented towards taking this approach.

**Possible Optimizations:**

* Currently to get the possible candidate pairs from our database we query it continuously, slowing down our operation in the process. To mitigate this we could use a structure like the k-vector (also known as an indexing vector) that’ll speed up the process
* We currently use DFS which is a recursive algorithm which however well it might do the job currently is still computationally expensive. One possible solution is to take an iterative approach to the graph traversal. More specifically an algorithm like A\* with a good heuristic and a priority queue could bring down the time complexity to O(nlogn), with n being the nodes explored.
* Weighing each vector pair in the QUEST algorithm to bring up reliability

**Approaches taken so far:**

* The first attempt was running Astrometry’s already implemented lost-in-space algorithm locally giving us a set of pixel coordinates for each image star and a set of equatorial coordinates for each catalog star, based on which the attitude could be determined. However this had the limitation of what resolution images we were allowed to use and for our intents and purposes it wouldn’t have been a viable solution
* Strict DFS: before implementing the subgraph scoring system I produced an assignment only if all image stars were given a catalog ID, which is not always possible
* Geometric hashing: to bring up the reliability of the algorithm I tried using a geometric hashing approach using triplets of stars (but not precomputing a database of triplets - just computing them after the initial pairwise filtering) and their area, angular distances and normalized side lengths to identify and filter out each triplet. However even after the initial pairwise filtering this approach was way too slow. The time complexity of computing the triplets was pretty much O(n^3). This could’ve been mitigated by precomputing a database of triplets (again, really slow) or using a quadtree to lower the complexity. However I decided not to take this approach.