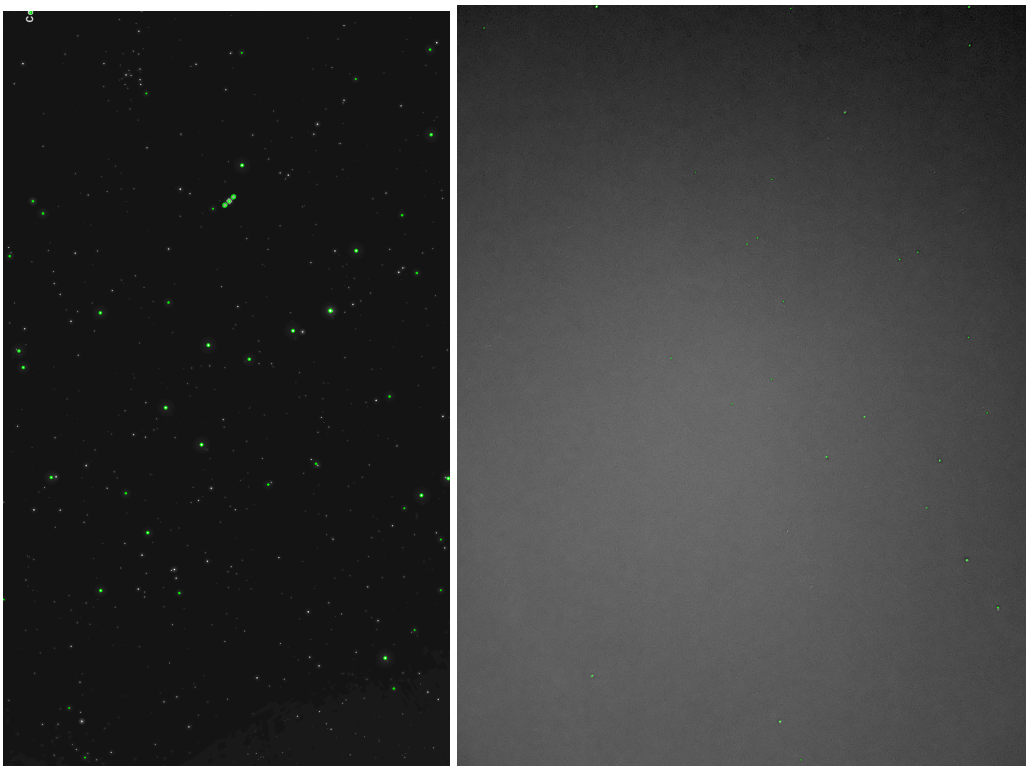
Part One: Algorithm - As Simple and Efficient as Possible

We begin with the assumption that we need to align only two images: one containing many hundreds of stars, and the other containing 10-20 stars. In this section, you are encouraged to explore existing algorithms and choose the one that seems simplest to implement.

Part One Answer: Implemented in StarsCords file

Step 1: Create a list of stars detected in the image, each with an ID, X, Y coordinates, radius, and brightness.
This is implemented using a **blob detector** in OpenCV, after applying pre-processing filters such as **TopHat** and **medianBlur**.



Step 2: Implemented in the imagesStarAlignment

Apply a transformation between image L1 and image L2. The goal is to align one image on top of the other. To do this, we need to estimate the following transformation parameters: **translation**, **rotation**, and **scale**:

(deltaX, deltaY, deltaAlpha, deltaScale) = T

How we do it:

1. Select 2 stars from L1 that are neither too close nor too far apart. In our case, we limited the minimum distance between them to 1000 pixels. We assume these two

stars also appear in image L2.

2. For all possible pairs in L2, compute the transformation T that maps the chosen L1 stars onto them.

3. Apply T to all stars in L1 to get L1T.

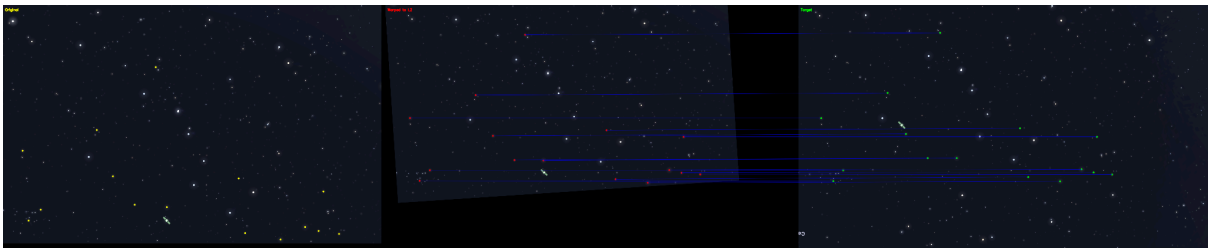
4. Count how many overlapping stars exist between L1T and L2 (within an EPSILON of 15 pixels).

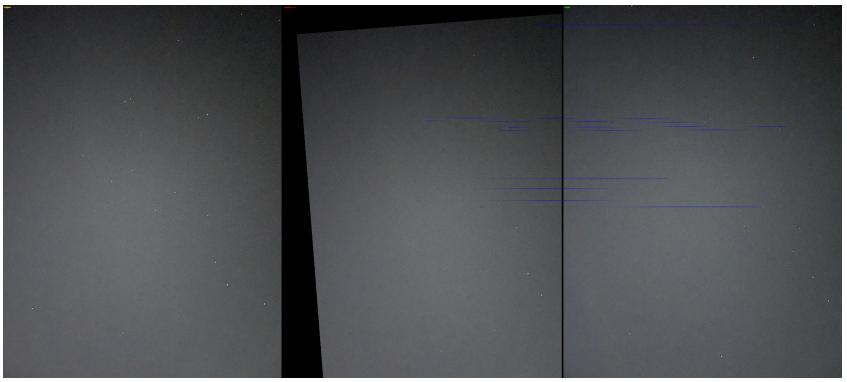
To reduce false positives (which are minimal thanks to the filters that prevent the blob detector from detecting too many fake stars), we repeat the algorithm for every pair among the top K brightest stars in L1.
The transformation T with the most overlapping stars is considered the correct match.

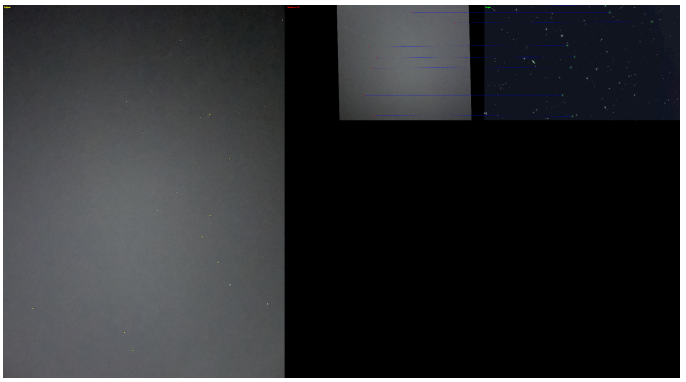
To prevent choosing stars in L1 that don't appear in L2 (e.g., stars that are actually planes or satellites), we define a threshold for the number of matching stars for T. For example, if a transformation T leads to only 5 matching stars, it is likely incorrect and will be discarded. If no satisfactory T is found, we declare a mismatch.

Output:

- A mapping of matching stars from the L1 list to L2.
- An image containing both the original images and the transformed image L1T.
- A visual overlay showing the matched stars and their locations in L1, L2, and L1T.





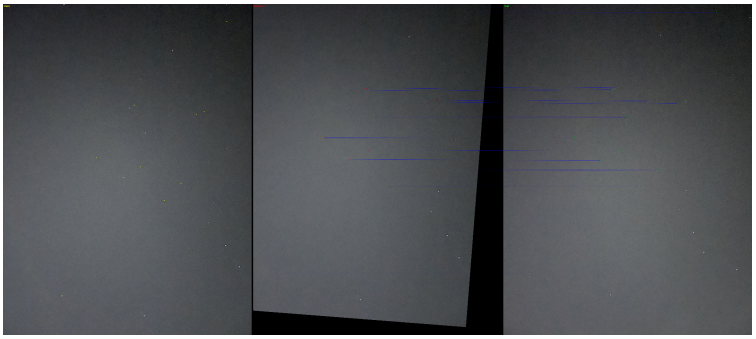


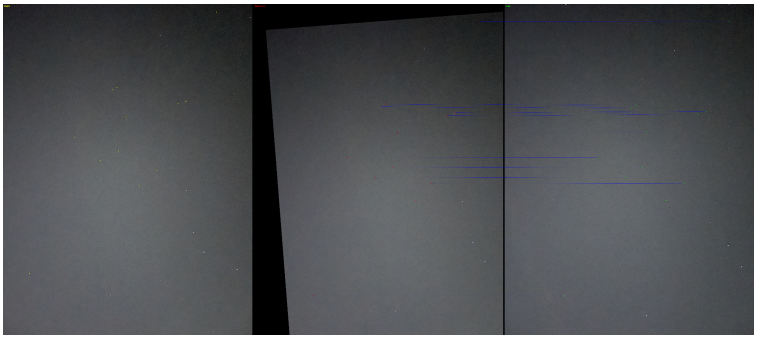
Alternative Approach: Implemented in a triangle match file
2. Use 3 stars instead of 2, starting from a certain minimum distance (e.g., 700 pixels). Compute the triangle angles between them and search for triplets in L2 with matching angles (within some epsilon tolerance).

- To make this efficient, precompute all triangle angles for every triplet of stars in L2 and store them in a hash table.
- Then find a matching triplet and compute the corresponding T.
- Use the T with the best matching result.

It's worth noting that this algorithm is significantly faster than the first one, but based on our results, it is generally less accurate.

Notes:
There are many ways to reduce the search space of triplets in L2. For example, filter by distance (not too large or small), know the camera's focal length, and search only within the field of view defined by the focal length.





Now lets see some more matching of different photo dataset: