

# Finding Pluto

(A project by Davide Vaiano and Costa Massena)

## Introduction

Pluto was discovered in 1930 by the astronomer Clyde Tombaugh, 23 at the time, before he even got his diploma.

After being hired by the Lowell Observatory just one year before, with the task of finding a rumored planet in the Kuiper Belt, Tombaugh had a brilliant idea that allowed him to discover what is now known as Pluto.

## How did he do it?

Clyde Tombaugh's job consisted of staring at pictures of specific star fields to find the differences, a tedious and time consuming task. He then had the idea of using those photographic plates combined with a blink comparator in order for the moving planet to appear clearer among the still stars.

The blink comparator, or blink microscope, is a device used by astronomers to detect moving objects in the sky.

It permits rapid switching from viewing one photograph to viewing the other, "blinking" back and forth between the two images taken of the same area of the sky at different times, giving us the impression of movement, and making it easier for us to see which part of the photo changed. Let's illustrate this with a small game. as you have noticed, we can see 2 similar images with 7 small differences.

Even though this one is quite easy, it takes your brain a bit of time to compare to side by side image.

We can, on the other hand, simulate our blink simulator by just switching between slides where the 2 images are aligned. Let's take a look :

See ? Way easier right ? And we can apply the same thing to a star field, as showed on these 2 slides

## Our objective

Our goal is to simulate the Pluto search using python.

We are given a set of photos taken one night a part of the exact same starfield, but those photos are not aligned, which would prevent the blinker from working properly.

The file contains 6 different photos of each night, with different color or brightness filters applied so that we can compare how fast we can detect the planet (what filter is the most efficient).

We can divide our projects into 2 big steps :

1) Making a python function that loads the images in specific files, and then a creates a new image from the second one which is aligned with the first one.

We will use keypoints to do so

2) After we have both files with aligned images, we need to create a second function that quickly switches from the night1 image to the night2 image with the same filter.

Costa :1) Aligning images

The images that we will compare are very similar, they only differ by the position of one light, Pluto.

Nevertheless, the images cannot be overlaid because they are not aligned.

So how do we align them ? We use keypoints

Keypoints are distinctive, identifiable points or features in an image that can be consistently detected across different views, lighting conditions, and transformations. They represent locations of interest that are stable and repeatable. By overlaying these points, we will align the images. For example take a look at these 2 pictures. They are not aligned, and not exactly the same ! Nevertheless, some points are identical to both photos, these are keypoints, if we align them, they will become aligned.

Let's take a look at the code. First the find best match function

This code implements a function that finds and visualises keypoint matches between two grayscale images using OpenCV.

The function employs the AKAZE (Accelerated-KAZE) feature detector and descriptor to identify distinctive keypoints in both images and compute their corresponding feature descriptors. It then uses a Brute-Force Matcher to find reliable correspondences between the keypoints of the two images.

The matches are sorted by their distance scores to prioritise the most similar feature pairs, and the function creates a visual representation by drawing lines connecting matched keypoints between the images.

It returns the sorted list of matches, the visualisation image, and the keypoints from both input images

Then the register image function

That geometrically aligns one image to match another using feature correspondences

The function extracts coordinate points from the matched keypoints identified in the previous matching step,

organising them into source and destination point arrays.

It then uses OpenCV's findHomography function with the RANSAC algorithm to robustly estimate a homography matrix that describes the geometric transformation needed to align the second image with the first,

Finally, the function applies this homography transformation using warpPerspective to geometrically warp the second image so that its features align with corresponding features in the first image.

## **BLINK COMPARATOR**

Now that our images are aligned, we can make our blink function.

So, basically, as I said before, blinking two images makes the differences between the two pictures taken of space reveal themselves more clearly. In our case, the only moving object in these photos does not represent a star, but rather, represents a planet, Pluto.

During the demonstration we will see how blinking between two images works, and it's definitely easier to notice the difference like that.

Let's take a look at the code:

It takes two images as input: typically the original image (img1) and the second image after it has been precisely registered (aligned) with the first (img\_registered).

There is a basic check about dimensions and type of pictures, cause if they have different sizes they blink anyway but there could be difficulty to understand better the differences.

If they have different type the code instantly return the function without blinking.

It then enters a loop, continuously displaying img1 for a very brief period, immediately followed by img\_registered for another brief period, and then repeats. This rapid alternation creates the "blinking" effect, crucial part for this project cause it makes notice the difference between first image and second image.

The loop continues until you press a key on your keyboard, allowing you to manually stop the blinking and observe any changes.

## CONCLUSION

This project learned us a lot about image processing, a concept we both weren't familiar with. Our biggest difficulties actually concerned small details, for example we started to work on Jupiter notebook, and it was complicated for both of us to upload the photos, which was quite frustrating because without that we couldn't begin the next steps. After that, we had a few issues understanding how to align the images from the key points, but once that was done, the rest of the coding went smoothly, and the time limit wasn't to much of a problem.