



Python studio

FINDING PLUTO

A project by Davide Vaiano and Costa Massena



INTRODUCTION





HOW WAS DISCOVERED?

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.



BLINK COMPARATOR





BLINK COMPARATOR





BLINK COMPARATOR





PLUTO STARFIELD



PLUTO STARFIELD



OUR OBJECTIVE

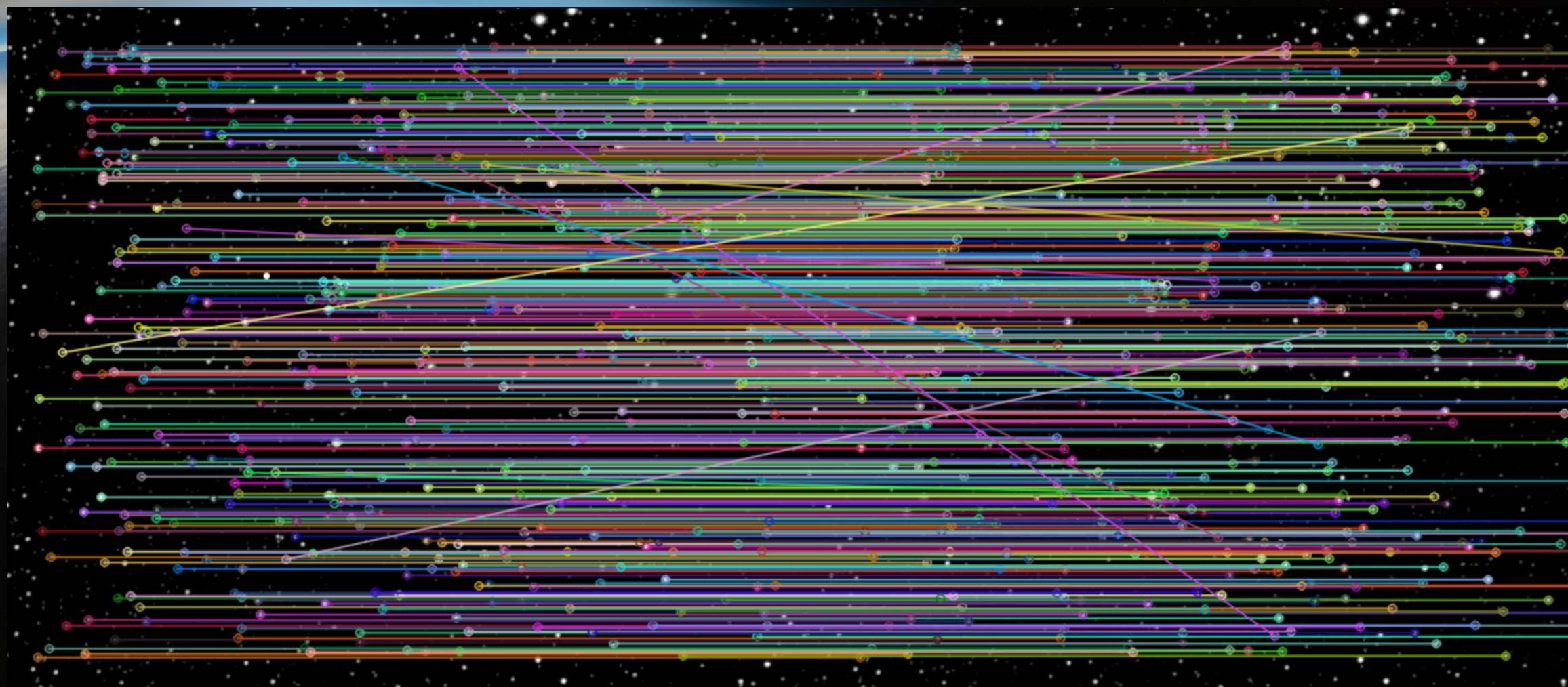
Use python to locate Pluto using images of the starfield

2 Steps :

- ★ Make a function to align 2 images using key points
- ★ Create a blinking function to rapidly switch between 2 images



ALIGNING IMAGES



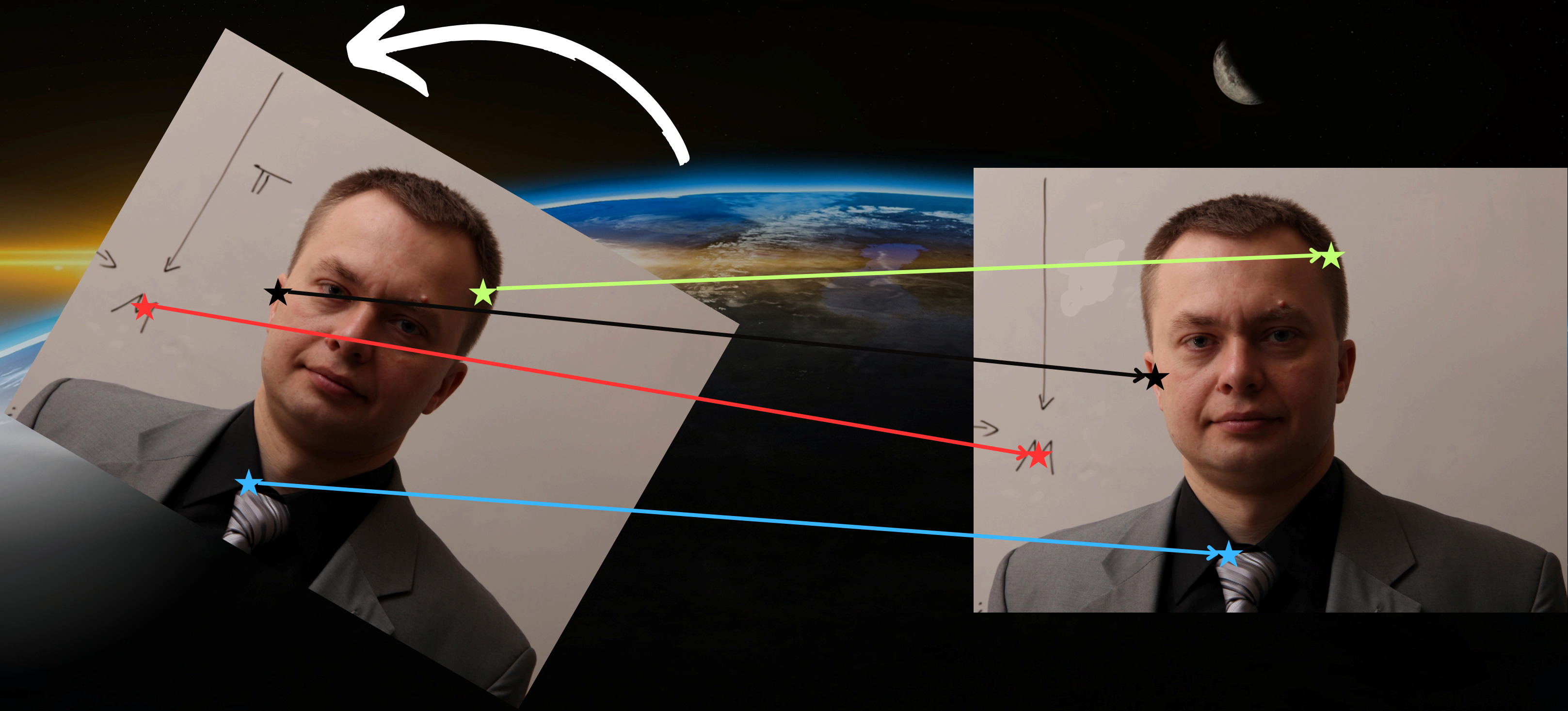


KEYPOINTS



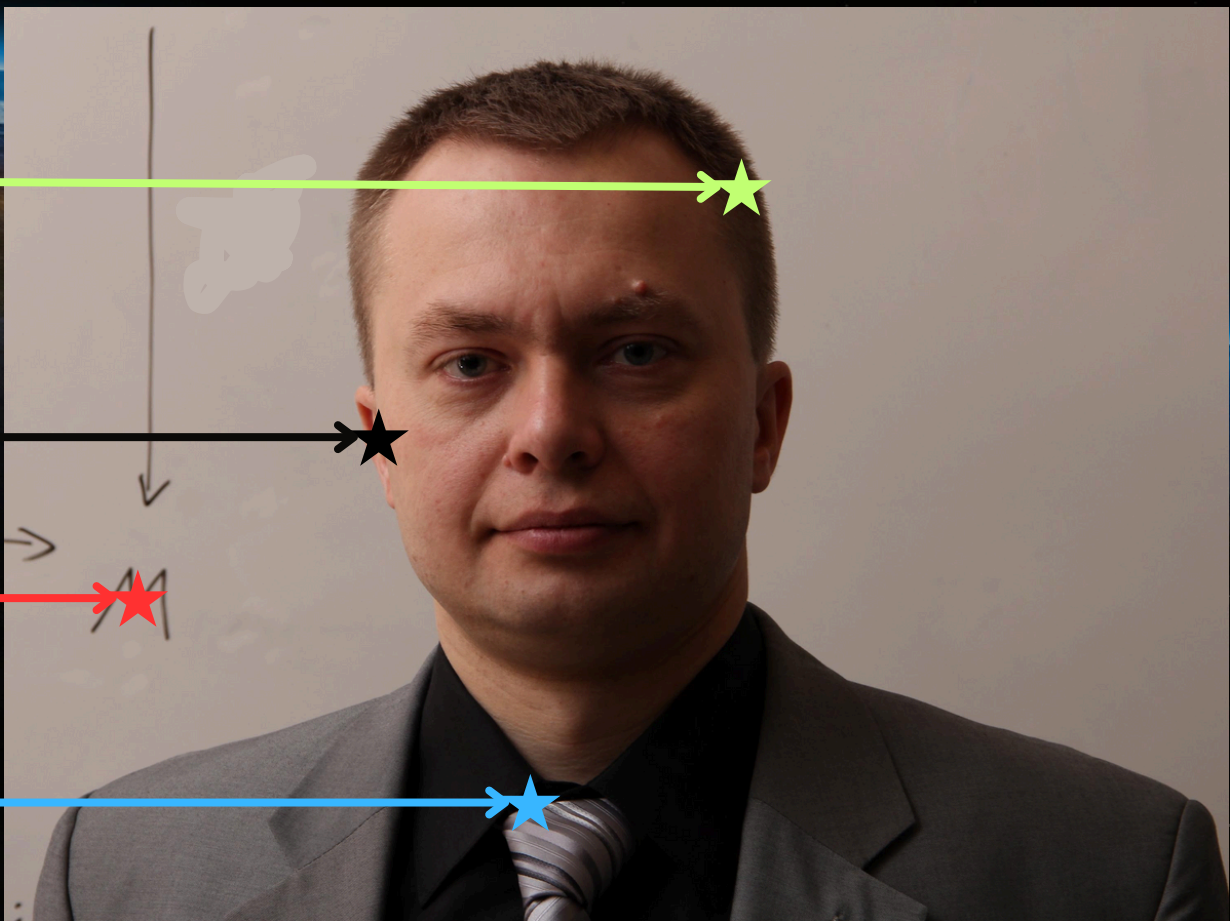
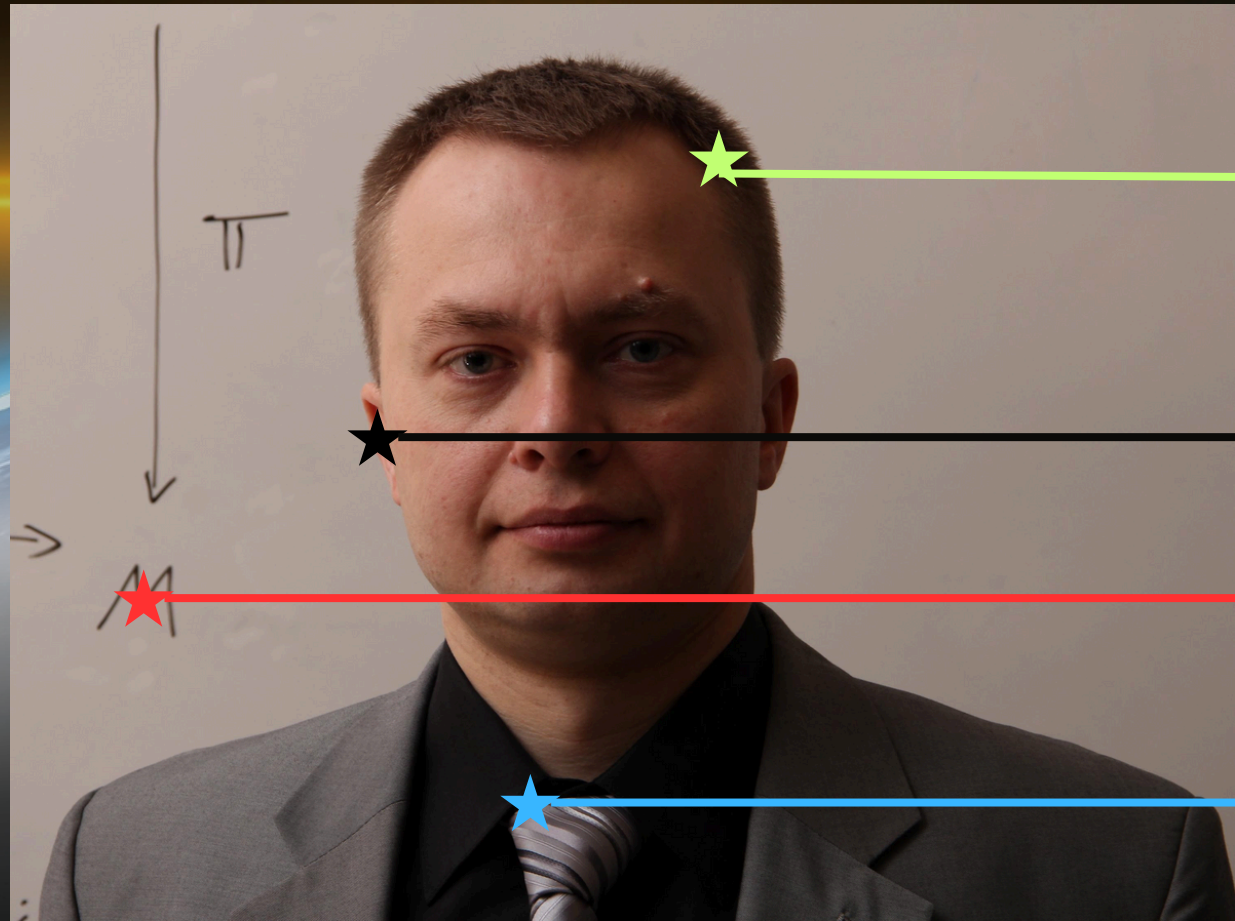


KEYPOINTS





KEYPOINTS





THE CODE:

```
# Function to find the best keypoint matches between two images
def find_best_matches(img1, img2):
    # Initialize the AKAZE detector/descriptor
    akaze = cv2.AKAZE_create()

    # Find keypoints and compute descriptors with AKAZE
    kp1, des1 = akaze.detectAndCompute(img1, None)
    kp2, des2 = akaze.detectAndCompute(img2, None)

    # Initialize the Brute-Force Matcher
    bf = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck=True) # Using crossCheck=True for more reliable matches

    # Find matches
    matches = bf.match(des1, des2)

    # Sort matches by distance (best matches are at the beginning)
    matches = sorted(matches, key=lambda x: x.distance)

    img_matches = None
    if len(matches) > 0:
        try:
            img_matches = cv2.drawMatches(img1, kp1, img2, kp2, matches, None, flags=cv2.DrawMatchesFlags_NOT_DRAW_SINGLE_POINTS)
        except TypeError as e:
            print(f"Error drawing matches: {e}. Check image type and depth.")
            img_matches = None # Ensure img_matches is None in case of error

    # Returns all sorted matches, the drawn image, and the keypoints
    return matches, img_matches, kp1, kp2
```

```
# Function to register (align) the second image to the first
def register_image(matches, kp1, kp2, img1, img2):
    """
    Registers img2 to img1 using matches and the homography matrix.

    Args:
        matches: List of best matches (cv2.DMatch objects).
        kp1: Keypoints of image 1.
        kp2: Keypoints of image 2.
        img2: The second image (grayscale) to be registered.

    Returns:
        A tuple containing:
        - The transformation matrix (homography H), or None in case of failure.
        - The registered (aligned) image, or None in case of failure.
    """
    # Extract corresponding points from keypoints
    src_pts = np.float32([kp1[m.queryIdx].pt for m in matches]).reshape(-1, 1, 2)
    dst_pts = np.float32([kp2[m.trainIdx].pt for m in matches]).reshape(-1, 1, 2)

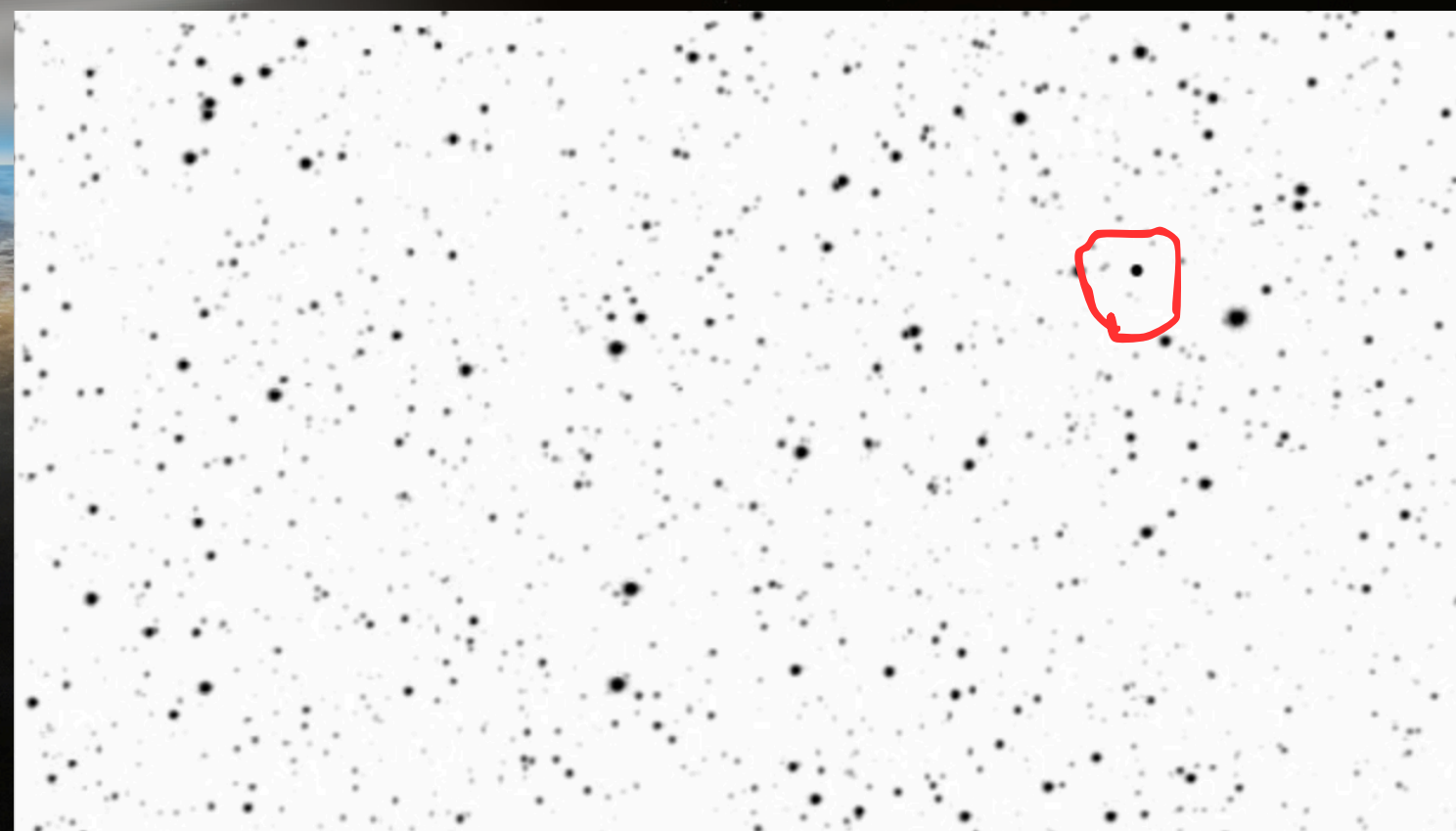
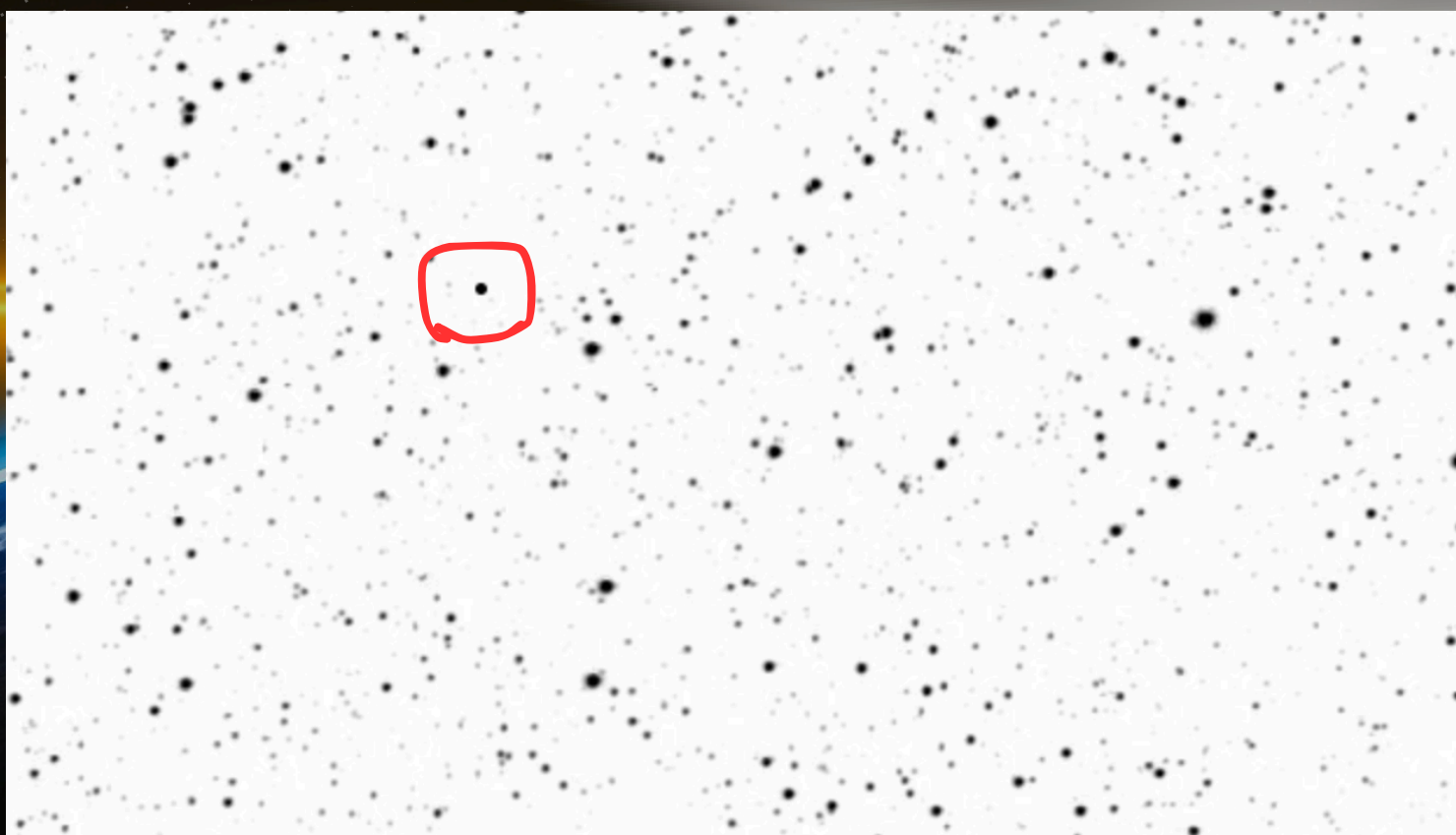
    # Find the homography matrix (the transformation to align img2 to img1)
    # RANSAC is a robust algorithm that handles outliers
    H, mask = cv2.findHomography(src_pts, dst_pts, cv2.RANSAC, 5.0)

    # Apply the transformation (homography) to the second image to align it with the first
    # The output dimension is the dimension of the first image (img1)
    height, width = img1.shape[:2] # We use the dimensions of img1 which is available in the main loop
    img_registered = cv2.warpPerspective(img2, H, (width, height))

    return H, img_registered
```



BLINK FUNCTION





THE CODE:

```
f blink(img1, img2, window_name="Blink Images", delay_ms=500):
    """
    Performs a blinking comparison between two images.
    Useful for spotting moving objects or differences between images.

    Args:
        img1: The first image (NumPy array).
        img2: The second image (NumPy array), ideally registered to the first.
        window_name: The name for the display window.
        delay_ms: Delay in milliseconds between switching images.
    """
    print(f"Starting 'blink' in window '{window_name}'. Press any key to stop.")

    # Check if images are valid and have the same dimensions
    if img1 is None or img2 is None:
        print("Error: One or both images for blinking are None.")
        return
    if img1.shape != img2.shape:
        print(f"Warning: Images for blinking have different dimensions ({img1.shape} vs {img2.shape}). Display may be inconsistent."
              # Consider resizing one image here if necessary,
              # but ideally img_registered should already have the same size as img1.

    while True:
        # Display the first image
        cv2.imshow(window_name, img1)
        # Wait for the specified delay or until a key is pressed
        key = cv2.waitKey(delay_ms)
        if key != -1: # If a key was pressed, exit the loop
            break

        # Display the second image
        cv2.imshow(window_name, img2)
        # Wait for the specified delay or until a key is pressed
        key = cv2.waitKey(delay_ms)
        if key != -1: # If a key was pressed, exit the loop
            break

    # Close the display window when the blink is stopped
    cv2.destroyAllWindows()
    print("'Blink' stopped.")
```



DEMONSTRATION



THANKS FOR YOUR ATTENTION

