Computer vision

Ruben Bosma

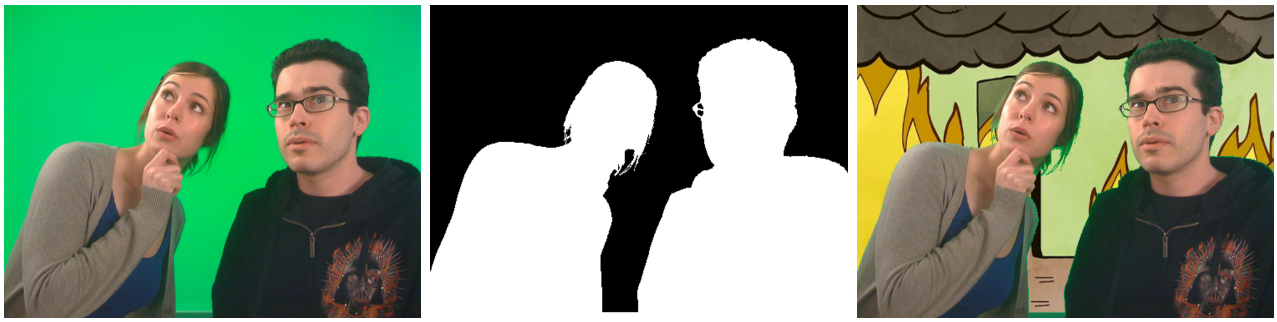
4 August 2025

Problem 1: Colour thresholding

To isolate the actors from the green background, pixel-wise thresholding is applied in the HSV colour space. A binary mask (Figure 1b) is generated by assigning a value of 255 to pixels where the hue, saturation, and value channels exceed the following lower bounds respectively: hue ≥ 68, saturation ≥ 100, and value ≥ 177. All other pixels are assigned a value of 0.

These threshold values were selected to preserve finer actor details while effectively removing most green background shades. The resulting binary mask is then inverted and used to extract the foreground (actors) from the original image (Figure 1a). The final composite image (Figure 1c) is produced by blending the extracted foreground with a new background image using simple per-pixel alpha blending.

While it performs well for most of the scene, some residual green pixels may remain along the actor boundaries. Further tightening the threshold can reduce these artifacts but risks discarding important visual features.



(a) Original image (b) Binary mask (c) Actors replaced

Figure 1: Green screen replacement using HSV thresholding and binary masking.

Problem 2: Image denoising with median filters

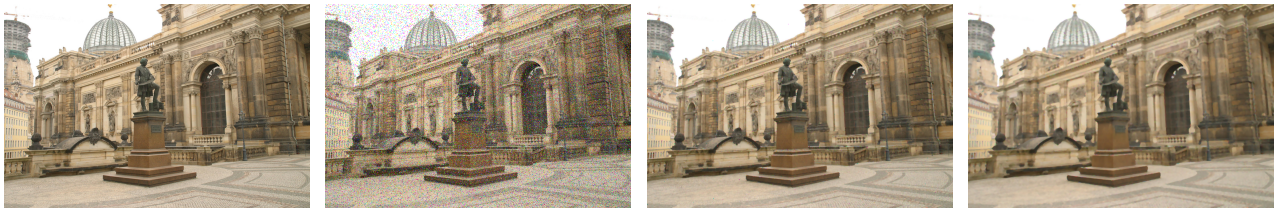
To simulate salt-and-pepper noise we applied a pixel-wise corruption algorithm. For a given density d, $d/2$ of the pixels were randomly set to 0 (black) and $d/2$ to 255 (white). Although the method was originally designed for greyscale images we applied it to a colour image Figure 2a, by corrupting each RGB channel independently.

As a result, corrupted pixels do not simply appear black or white, but instead show bright, random colours. This happens because different channels are affected independently.

To remove the noise, a median filter was manually applied to each channel using a filter of size $k \times k$, with $k \in \{3, 5, 7\}$. Results are shown in Figure 2 for two noise levels:

At $d = 0.1$ (top row of Figure 2b–d), the corruption was relatively mild and the median filter performed well. Only minor artefacts remained. The 3×3 filter helped remove most of the noise without major blurring, whilst 5×5 removed even more noise the blur effect increased. (Figure 2c).

At $d = 0.2$ (bottom row of Figure 2e–h), the noise was more severe, denoising became less effective and coloured pixels remained more prominently, for this density the 5×5 provided the best balance between visual clarity and noise removal.



(a) Original (b) Noisy ($d = 0.1$) (c) Filtered 3×3 (d) Filtered 5×5

(e) Noisy ($d = 0.2$) (f) Filtered 3×3 (g) Filtered 5×5 (h) Filtered 7×7

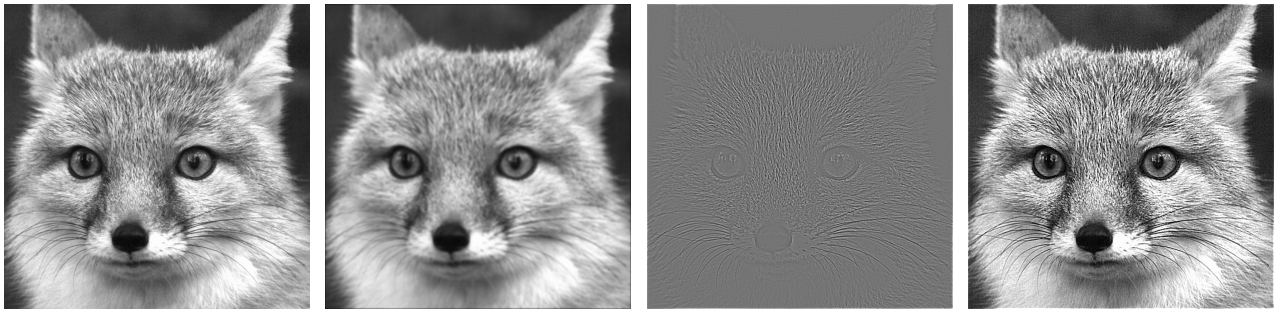
Figure 2: Salt-and-pepper noise simulation and denoising on `semper1.jpg`. Top row (b–d): $d = 0.1$; Bottom row (e–h): $d = 0.2$.

Problem 3: Image sharpening through unsharp masking

To enhance edge detail in a greyscale image, unsharp masking was applied to the original image (Figure 3a). The process involved first converting the colour image to greyscale, then computing a blurred version using a guassian filter of size 5×5 and standard deviation $\sigma = 2.0$ (Figure 3b). A mask was computed as the pixel-wise difference between the original and the blurred image. This mask was then scaled to the range $[0, 255]$ for visualisation purposes (Figure 3c).
The sharpened result (Figure 3d) was obtained by adding a scaled version of the mask back to the original image, using the formula:

$$\text{Sharpened} = \text{Original} + \alpha \cdot \text{Mask}$$

with $\alpha = 1.5$. This value was chosen after experimentation as it produced a visually appealing sharpness without introducing artefacts. Increasing α further would sharpen edges more aggressively.

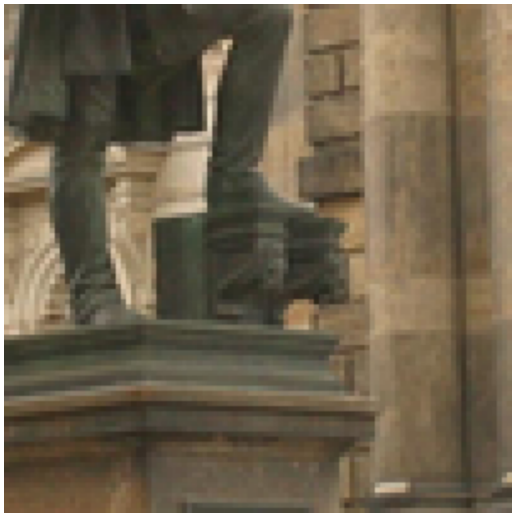


(a) Original (b) Blurred (c) Mask (d) Sharpened

Figure 3: Unsharp masking process. The mask was scaled to the $[0, 255]$ range for visualisation.

Problem 4: Bilinear interpolation for image resizing

A function was implemented to resize a colour image by a given scale factor s, using both nearest-neighbour (NN) and bilinear (BL) interpolation. Each method was applied independently, and results were generated for several values of s, including a clear close up comparison at $s = 2.0$.
Nearest-neighbour interpolation selects the closest pixel value without considering any neighbouring intensity differences, resulting in sharp but blocky outputs. Bilinear interpolation computed a weighted average of the surrounding four pixels and produced smoother results by approximating intermediate values.
To illustrate these differences, both methods were applied to upscale the same region of a high-resolution image. As shown Figure 4, the nearest-neighbour version in Figure 4a appears pixelated and jagged, particularly along edges and fine details, whereas bilinear interpolation in Figure 4b produced smoother edges and overall better looking results.



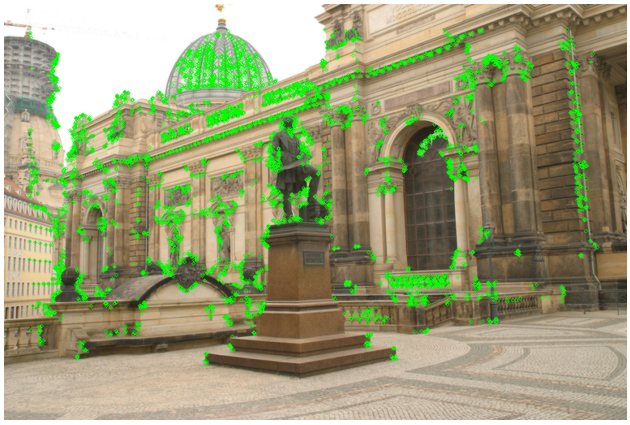
(a) NN close-up $s = 2.0$

(b) BL close-up $s = 2.0$

Figure 4: Close-up comparison of interpolation methods for upscaling by $s = 2.0$.

Problem 5: Feature Detection and Matching

The ORB algorithm (Rublee et al., 2011) was used for keypoint detection and description. ORB combines FAST detection with BRIEF descriptors, adding rotation invariance and multi-scale analysis. It is efficient, free from patent restrictions, and available in OpenCV.
Feature matching was performed using brute-force Hamming distance with cross-checking enabled to filter inconsistent matches. Only the top 100 matches (by descriptor distance) were visualised to improve clarity and highlight meaningful correspondences.



(a) Keypoints in image A

(b) Keypoints in image B

(c) Top 100 ORB matches

Figure 5: ORB keypoint detection and matching using brute-force Hamming distance.

A small grid search was conducted to evaluate `nfeatures` and `nlevels`, using average RANSAC inliers (Bala, 2015) as the metric. As shown in Figure 6.

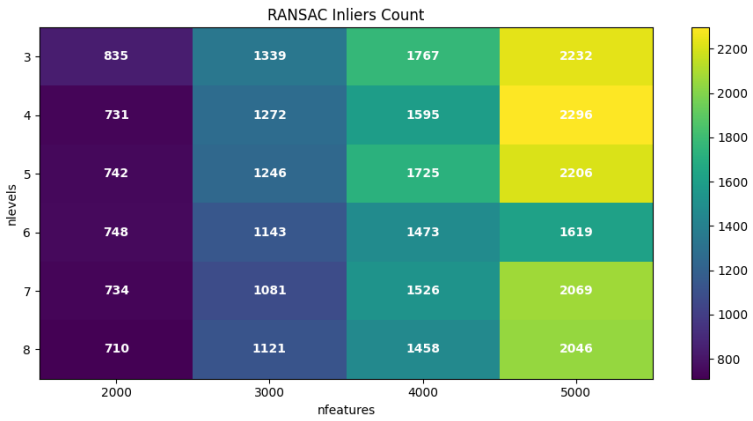


Figure 6: Effect of ORB parameters on RANSAC inliers.

The final parameters were:

- nfeatures = 5000: enough keypoints without redundancy,
- nlevels = 4: optimal pyramid depth for scale detection,
- scaleFactor = 1.2: moderate step size across scales.

Problem 6: Feature Matching Accuracy Against Scale Change

This experiment evaluates how well ORB feature matching handles scale variation. The same parameters from Problem 5 were used: nfeatures = 5000, scaleFactor = 1.2, and nlevels = 4. Descriptors were matched using a brute-force Hamming matcher with cross-checking.
A test image A was scaled across a range of factors from 0.1 to 3.0 (step size = 0.1), producing image B. Keypoints were extracted and matched between A and B. Accuracy was measured as the proportion of matches whose transformed coordinates fell within a 2-pixel tolerance of the expected location in B.

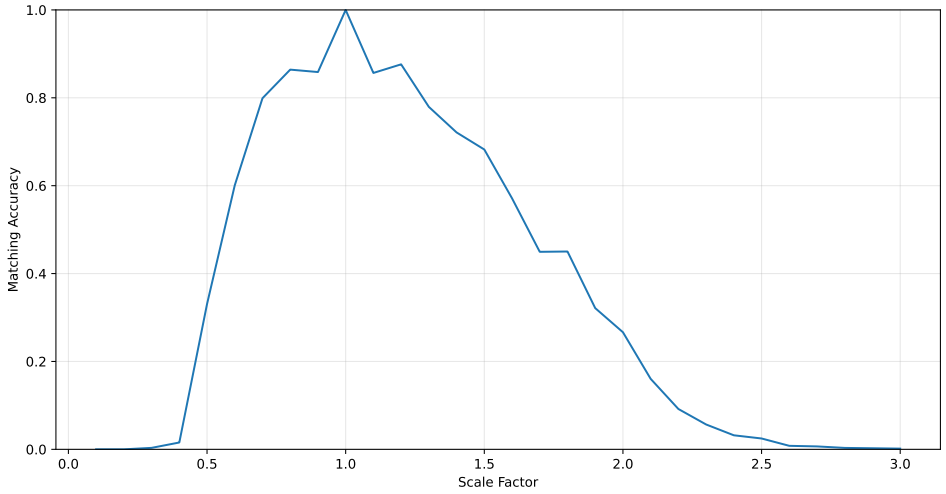


Figure 7: Feature matching accuracy vs scale factor. Matching peaks near scale 1.0 and degrades at larger and smaller scales.

As seen in Figure 7, accuracy is highest near the original scale (1.0), reaching 100%. It gradually decreases for both upscaling and downscaling. At a scale factor of 0.5, accuracy drops to approximately 30%.

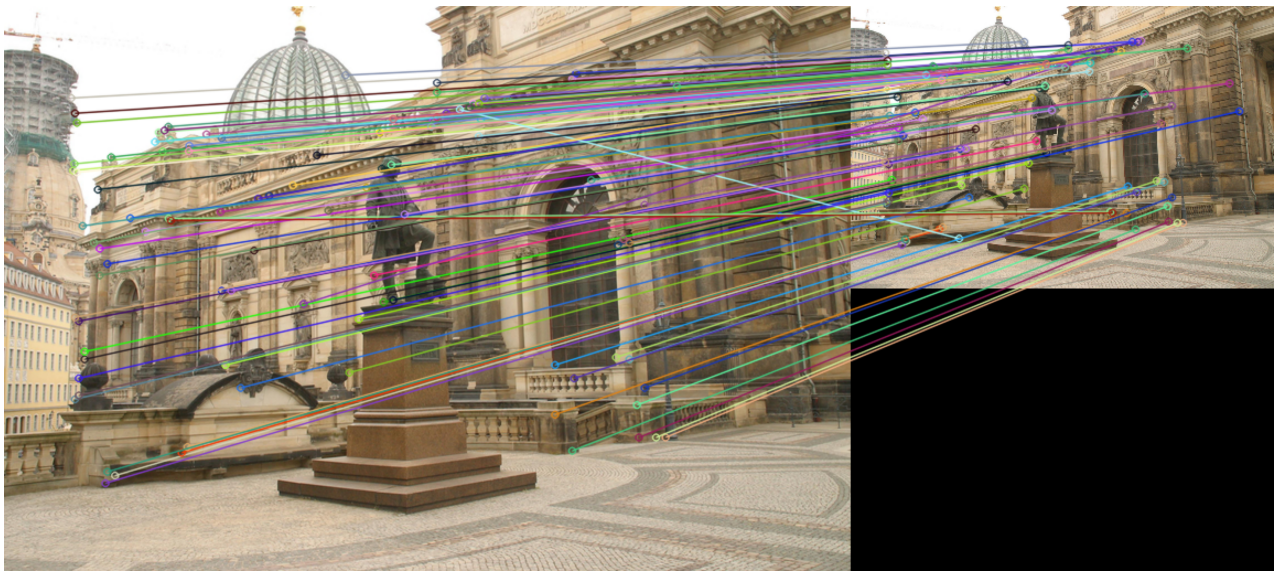


Figure 8: Top 100 ORB matches between original and scaled (0.5×) images.

Figure 8 shows that with 100 strongest matches, many correspondences remain but we can also see inaccurate matches with loss of scale precision. These results confirm that ORB is reasonably stable with scaling within the range $0.75 \leq s \leq 1.25$, but performance degrades for more extreme scaling.

References

Kavita Bala. Cs4670/5760: Computer vision – lecture 13: Ransac. https://www.cs.cornell.edu/courses/cs4670/2015sp/lectures/lec13$_{r}ansac_{w}eb.pdf$, 2015. *Accessed* : $2025 - 07 - 30$.

Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. Orb: an efficient alternative to sift or surf. pages 2564–2571, 11 2011. doi: 10.1109/ICCV.2011.6126544.