

inkmap — methods and results

- [inkmap — methods and results](#)
 - [1. The problem](#)
 - [2. Approach: the model is the input](#)
 - [3. Terminology](#)
 - [4. Locating the mark](#)
 - [5. Refining the transform](#)
 - [6. Coverage, and the three 0–1 channels](#)
 - [7. Validation strategy](#)
 - [8. Results](#)
 - [9. Limits](#)
 - [10. Reproducing everything](#)
 - [Datasets used](#)
 - [References](#)

inkmap — methods and results

Per-pixel foreground coverage for photographed marks.

Given a photograph of a printed mark and a *model* of its shape, inkmap reports, for every pixel, the fraction of that pixel's area covered by the mark. Not a boolean mask: a continuous 0–1 area fraction, accurate enough to make physical measurements on.

This document is the approach, the algorithms, and the evidence — including the places where the evidence is weak, and one defect the evaluation found and forced a fix for.

1. The problem

Decoding a QR code is solved. Reading text is solved. **Measuring the mark itself** is not. If you need the reflectance, density, or colour *of the printed material* — as opposed to what it says — you need to know exactly which pixels are mark, which are substrate, and which are neither.

Three things make this harder than it looks:

1. **A boolean mask is not enough.** A pixel straddling the mark's edge is genuinely part ink and part substrate. Forcing it to one class biases any average computed over the mask.
2. **Geometric purity is not sufficient either.** The imaging blur drags substrate signal into pixels that lie *entirely* under ink. This contamination has a different spatial footprint from partial area coverage, so no coverage-based weighting removes it (§6, §8.4).

3. **Thresholding puts its decision boundary in the worst place.** Every binarisation method places its boundary in the middle of the blurred edge — exactly where a photometric measurement must not be taken.

2. Approach: the model is the input

The central design decision is that **the shape is a parameter, not a special case**. A model answers exactly one question:

Given a point in the mark's own coordinate system, is that point foreground?

```
class Model(Protocol):
    extent: tuple[float, float]      # size of model space, in the model's own units
    feature_size: float              # smallest feature that must be resolved
    def occupancy(self, u, v): ...   # vectorised foreground indicator
```

Everything else — locating, sub-pixel refinement, rasterising, measuring — is written against that one question. So QR symbols, linear barcodes, printed letters and arbitrary artwork all traverse one pipeline, and a user-supplied shape needs two members and no base class.

Shipped models:

Model	Foreground defined by	Exact?
MatrixModel	a grid of cells (QR, Data Matrix, Aztec)	yes — analytic lookup
BarsModel	a run of bars (EAN, UPC, Code 128, Code 39, ITF)	yes — analytic lookup
PolygonModel	closed outlines, holes by even-odd	yes — point-in-polygon
RasterModel	a bitmap (logo, scan, pre-rendered glyph)	to raster resolution
GlyphModel	printed text, rendered via Pillow	to raster resolution

Pipeline

```
photograph → locate → recover model → refine transform → rasterise → erode
              (§4)      (§4.2)          (§5)                (§6)      (§6.3)
```

3. Terminology

Classes are **foreground** and **background**, never dark/light. Appearance inverts — white-on-black print, reflectance-reversed symbols (explicitly permitted by ISO/IEC 18004), inverted imaging modalities — so an appearance-based name would mean the mark in one image and the substrate in the next. **Polarity** carries that mapping explicitly.

`BACKGROUND` is scoped to *within* the mark’s footprint; the specified clear surround is `MARGIN` and everything else is `OUTSIDE`, so the ambiguity that makes a bare “background” dangerous cannot arise.

The continuous value is **coverage**, after ISO 12647-2:2013 §3.1 *area coverage* — “ratio of the area covered with ink to the entire area” — the only term in any standard whose definition is literally the quantity computed. Full cross-field mapping (ISO/IEC 18004, 15415, DIBCO, matting, remote sensing, medical imaging) is in [terminology.md](#).

4. Locating the mark

4.1 A cascade, not a detector

Measured on 189 real photographs from Wikimedia Commons:

locator	images with a detection
zxing-cpp	74%
OpenCV <code>QRCodeDetectorAruco</code>	68%
OpenCV <code>QRCodeDetector</code>	43%
cascade (union)	80%

Using OpenCV’s classic detector alone forfeits ~37 points of recall. All three are permissively licensed (Apache-2.0) pure wheels. Deliberately **not** used: `qrdet` / `qreader`, which are MIT on the tin but depend on `ultralitics`, which is AGPL-3.0.

For marks nothing can auto-detect — text, logos, engraved marks — `inkmap.place(model, quad)` takes the corners from the caller and the rest of the pipeline is identical.

4.2 Recovering the model, and one thing never to do

The model is read **from the image**: OpenCV’s `straight_qrcode` gives a bit-exact rectified matrix where available, otherwise the grid is sampled directly at cell centres (the approach ISO/IEC 18004 clause 12 specifies for decoding).

Never by re-encoding the decoded payload. Error-correction level, mask pattern, and Code 128 code-set switching are encoder choices that the decoded string does not pin down, so re-encoding can silently produce a different-sized, differently-masked pattern. It is a silent-wrong-answer path.

4.3 The corner-order trap

Locator corners are used in the **model’s own order** (the corner corresponding to model `(0,0)` first), never re-sorted into image order. Sorting corners by coordinate sum/difference is a widespread idiom; it works until the mark rotates past 45°, at which point it permutes the corners relative to the model — rotating the model and producing a mask of exactly the right size and density in the wrong places.

This is regression-tested at eight rotation angles precisely because it is invisible at 0°.

5. Refining the transform

Four locator corners are four observations; the model has thousands. `inkmap` renders the model at the observed scale and aligns it to the photograph by ECC, optimising the full 8-DoF homography.

condition	locator corners (px RMS)	refined (px RMS)	gain
clean, frontal	0.87	0.02	42×
mild perspective	1.05	0.02	53×
strong perspective	1.08	0.05	23×
defocused ($\sigma=3$)	0.99	0.08	12×
noisy + gradient	1.31	0.10	14×
glare + JPEG q70	0.90	0.21	4×

Locators return integer corner coordinates, so ~ 1 px is their floor regardless of image quality.

5.1 Three half-pixel conventions, and 50× of accuracy

Each of these is invisible on inspection — the mask still looks like a mask:

1. **Pixel centre vs pixel corner.** OpenCV treats integer coordinate `x` as the *centre* of a pixel spanning `[x-0.5, x+0.5)`. Rasterising `[x, x+1)` instead — the natural array-indexing reading — offsets the entire coverage map by half a pixel.
2. **The template raster.** A model rendered at `s` px/unit puts template pixel `t` over `[t/s, (t+1)/s)` in model units, so its centre is `(t+0.5)/s`. Omitting that half-pixel injects a fixed 0.71 px bias into every refinement.
3. **Locators disagree with each other.** OpenCV's QR corners are the integer index of the last *contained* pixel (span `N·s - 1`); `zxing-cpp` returns the continuous outer boundary (span `N·s`). Measured directly, not read from documentation.

The diagnostic worth remembering: if refinement error grows with the mark's rotation angle following $R(d) \sim d$, the residual `d` is a coordinate-convention offset, not noise. That signature is what located all three.

A fourth, related trap: the ECC template needs a rendered clear margin. Blurring a template that stops at the mark's edge makes the kernel reflect interior content across the border, inventing an edge profile the photograph does not have — ECC then shifts the whole fit to match the artefact (~ 0.45 px).

6. Coverage, and the three 0–1 channels

6.1 Rasterisation

Each pixel is probed at `samples2` sub-positions on a regular grid at sub-pixel centres; each probe is back-projected through the transform and passed to `Model.occupancy`. The mean is the covered area

fraction — ordinary supersampled area coverage. Only the model’s bounding box is evaluated, so a mark occupying 2% of a 12 MP photo costs 2% of the work.

6.2 Three different questions

Channel	What it is	Use it for
coverage	geometric — fraction of the pixel’s area under foreground	reporting, print QC, finding mixed pixels
weight	blur-aware — coverage eroded by how far the PSF reaches	the only channel that may gate a measurement
confidence	epistemic — trust in the label, given clipping, glare, model/image disagreement	quality gating, artefact rejection

If the underlying reality is binary and we are merely unsure which class a pixel is, the “probability of foreground” is `confidence`. If reality is genuinely not binary because the pixel straddles an edge, the degree of membership is `coverage`. Both are real numbers about different things: a pixel can be exactly half foreground (`coverage` 0.5) and we can be completely certain of that (`confidence` 1.0).

6.3 Erosion, and why coverage alone cannot fix it

`weight` is grayscale morphological erosion of the coverage map — exactly “the minimum coverage within radius r ” — with radius $\approx 2\sigma$ of the estimated blur.

7. Validation strategy

Three tiers, because no single one is sufficient:

Tier	Question	Ground truth
Synthetic	Is the coverage value <i>correct</i> ?	exact, by construction
Real photographs	Is the geometry right on real images?	an independent decoder
DIBCO	Does the contamination claim hold on <i>real ink</i> ?	hand-labelled per-pixel masks

7.1 Why synthetic scenes are unavoidable

`synth.render` builds the image **from** the coverage map: supersample the model through a known transform, then form the image as `255 · (1 - coverage)`. Ground truth and stimulus are the same object by construction, so degradations applied afterwards (blur, noise, illumination gradient, glare, JPEG) leave the truth exactly correct.

This is not a convenience. **No public dataset has per-pixel ink ground truth for QR codes or barcodes** — a survey of ArTe-Lab, Muenster BarcodeDB, BoofCV’s QR benchmark, InventBar/ParcelBar and BarBeR found that every published “mask” is a *solid filled quadrilateral* over the whole symbol footprint, not per-bar or per-module ink. This was verified by opening the masks and counting

foreground runs per scanline: 1.00–1.04 runs per row, where per-bar ink would give many. Region ground truth cannot validate a coverage map.

7.2 The round-trip test, and its calibrated sensitivity

For real photographs, no per-pixel truth exists — so instead: sample the model through inkmap’s *own* fitted transform, re-render it cleanly, and hand it to a decoder. The decoder took no part in fitting the geometry and has error correction, so it does not accept near-misses.

Sensitivity, measured by deliberately perturbing a known-good transform:

transform error	round-trip
0.00 units	pass
0.10 units	pass
0.25 units	pass
0.50 units	fail
0.75 units	fail

So it certifies “within a quarter of a unit”, not “perfect”. It is reported as such.

8. Results

8.1 Coverage accuracy (synthetic, exact ground truth)

Mean absolute error against exact truth, at 9.7 px/unit:

condition	overall MAE	MAE on mixed pixels
clean	0.0000	0.000
mild perspective	0.0000	0.003
strong perspective	0.0001	0.009
glare + JPEG q70	0.0005	0.042

Error accumulates only on mixed (edge) pixels, which is consistent: coverage error is dominated by residual transform error, and only edge pixels are sensitive to it.

Across model kinds, same conditions — the pipeline does not care what shape it is given:

model	coverage MAE
MatrixModel (QR)	0.00004
RasterModel	0.00001
PolygonModel	0.00008
BarsModel	0.00013

model	coverage MAE
GlyphModel	0.00004

8.2 Real photographs: QR

189 CC-licensed Commons photographs (signs, posters, packaging, business cards, stickers, billboards, menus, tickets, screens), downsampled to 1600 px.

metric	value
images with ≥ 1 mark	152 / 189 (80%)
marks found	192
round-trip decode	77% (n=177)
round-trip among <i>measurable</i> marks	96% (n=93)
module agreement vs OpenCV's independent rectification	0.999 median
IoU vs independent Sauvola binarisation	0.871 median
foreground darker than background	176/176 (100%), median separation 156 grey levels
judged measurable	51%
median runtime	252 ms

Round-trip against resolution:

px/feature	n	round-trip	bin. IoU	measurable
0–3	21	24%	0.671	0%
3–4	31	74%	0.866	35%
4–6	52	79%	0.890	65%
6–10	52	92%	0.906	67%
≥ 10	21	90%	0.917	62%

The headline is the conditional: when inkmap judges an image measurable, the geometry is independently confirmed correct 96% of the time. The failures are concentrated where the library already says not to trust it.

8.3 Real photographs: barcodes — a defect the evaluation caught

56 Commons photographs containing linear barcodes.

The **first** run scored a round-trip rate of 5.3%, against 77% for QR. That gap was too large to be resolution alone, and it was not: `profile_to_bars` estimated the module width from a percentile of run lengths, and returned **105 modules for an EAN-13, whose specification fixes it at exactly 95**. Every bar after the first was then placed wrong — while the mask still looked like a barcode.

Two fixes:

1. **Use the specification where it pins the count.** EAN-13/UPC-A = 95, EAN-8 = 67, UPC-E = 51. Estimating a number the standard already fixes is a needless risk.
2. **Verify variable-length symbologies against the payload.** Code 128, Code 39 and ITF have no fixed count, so the fitted pattern is re-rendered and decoded; if it does not return the payload the decoder already read, neighbouring module counts are tried. This is not circular — the payload came from reading the photograph, the pattern is reconstructed from our own geometry, and agreement is evidence the geometry is right.

Result on synthetic barcodes with known ground truth:

symbology	truth	before	after
EAN-13	95	105 ✗	95 ✓
EAN-8	67	—	67 ✓
UPC-A	95	—	95 ✓
Code 128	156	155 ✗	156 ✓
Code 39	143	143 ✓	143 ✓
ITF	—	✗	✗ still fails

And on the real corpus: **round-trip 5.3% → 47.5%**.

metric	value
images with ≥ 1 barcode	53 / 56 (95%)
round-trip decode	47.5% (n=61)
round-trip among <i>measurable</i> marks	89% (n=18)
IoU vs Sauvola	0.819 median
foreground darker than background	100%
median px/feature	3.6
judged measurable	30%

Barcode support is weaker than QR and should be treated as such. Median resolution is 3.6 px per narrow bar — most photographs of barcodes in the wild simply do not resolve the narrowest element. ITF still fails round-trip.

8.4 Real ink: the contamination claim, tested against hand-labelled truth

Everything in §6.3 was established on synthetic scenes with a blur we applied ourselves. DIBCO (Document Image Binarization Contest, 2009–2019) is the only public corpus with exact per-pixel ink ground truth. It is documents, not marks, so inkmap’s *modelled* path does not apply — but it can answer the question that matters: **does a geometrically perfect mask really carry contamination on real ink?**

Method: take the hand-labelled GT foreground, erode it by r pixels, measure the mean grey level. If boundary pixels were clean, the mean would not move.

46 matched image/GT pairs:

erosion r	foreground mean	shift vs r=0	as % of contrast	pixels kept
0	108.8	—	—	100%
1	95.6	−13.2	−16.8%	63.5%
2	88.2	−20.6	−26.2%	19.2%
3	76.7	−32.0	−40.9%	5.6%
4	84.6	−24.2	−30.8%	2.8%

Median background 200.9; median foreground/background contrast 78.4 grey levels.

A perfect, hand-labelled mask carries 17% of the contrast as boundary contamination at one pixel of erosion, 26% at two. The claim is not an artefact of the synthetic generator — inkmap played no part in producing these masks. (r=4 reverses because only 2.8% of pixels survive; treat $r \geq 3$ as sample-starved on this corpus, whose documents are mostly low-resolution historical scans.)

8.5 Photometric bias (synthetic, exact truth)

Recovering a foreground-only signal (true value 100) from a frame blurred by the camera PSF:

px/feature	PSF σ	coverage > 0.5	coverage == 1.0	weight (2σ)
12	1.0	−6.61%	−4.10%	−0.07%
12	2.0	−12.87%	−10.47%	−0.37%
12	3.0	−18.86%	−16.76%	−0.49%
6	1.0	−12.89%	−8.53%	−0.26%
6	2.0	−24.40%	−21.05%	−0.63%
6	3.0	−33.78%	−31.52%	−1.40%

A thresholded mask — what any binarisation gives you — is biased by up to 34%. Geometrically pure pixels still carry up to 32%. Erosion by $\sim 2\sigma$ holds it below 1.5%.

8.6 Blur estimation

true σ	estimated (median)
0.5	0.30
1.0	0.92
1.5	1.54
2.0	2.16
3.0	2.78

Accurate between ~ 1 and ~ 2 px. A minimum at the top of the search range is reported as `nan` rather than as a measurement: it means the modelled shape and the photograph do not correspond, and reporting the boundary value would silently size the measurement margin from noise.

8.7 Text: the honest negative

Text is where inkmap is weakest, and the evaluation says exactly why.

86 words from 139 real photographs, located by Tesseract (an independent oracle for *what* and *where*), then modelled with `GlyphModel`:

metric	value
glyph model vs independent binarisation (IoU)	0.173
image-derived raster model, same placement (IoU)	0.756
words with ≥ 6 px/feature	9 / 86
judged measurable	5%

The gap between 0.173 and 0.756 is the whole story: **placement works; the glyph shape is wrong**. A raster model taken from the image has the true shape by construction, and it scores 0.76 under identical placement.

The cause is the font, confirmed directly. Text was rendered in a known font and modelled with each of four fonts; the cell is IoU against exact truth:

truth \ model	Helvetica	Times	Courier	Georgia
Helvetica	0.999	0.593	0.463	0.593
Times	0.591	0.997	0.511	0.797
Courier	0.457	0.511	1.000	0.517
Georgia	0.589	0.799	0.523	0.998

With the correct font, IoU is 0.997–1.000. With the wrong one, 0.46–0.80. inkmap’s modelled path for text is therefore only as good as the font you give it.

Practical guidance: for text, either pass the actual font (`models.text_model(word, font_path=...)`), or use a `RasterModel / source="observed"` path that takes the shape from the image. And note that most real-world text is simply too small — only 9 of 86 words reached 6 px per stroke width.

9. Limits

- **No text locator.** `GlyphModel` models text but nothing here finds it; pass `quad=` from your own OCR or layout analysis.
- **Barcodes are weaker than QR** (§8.3), and ITF round-trip still fails.
- **Micro QR and rMQR are ignored**, not fitted — different module counts.
- **Non-planar substrates** (bottles, wrinkled labels) degrade toward the edges: one homography assumes a plane. QR alignment patterns (version ≥ 2) would support piecewise correction; not implemented.

- **Halftoned/screened print:** modelled coverage reports *nominal* geometry. If the material is a halftone screen rather than solid fill, `source="observed"` reflects reality and their disagreement is the diagnostic.
- **Coverage is geometric, not optical.** It says where foreground *should* be; it cannot see material that failed to deposit. Reflectance-derived coverage is inflated by lateral scattering in the substrate (Yule–Nielsen), which is why the observed path is named *apparent* coverage.
- **DIBCO evidence is about contamination, not about inkmap’s masks** — inkmap does not produce the masks used in §8.4, and is not a document binariser.

10. Reproducing everything

```
python scripts/validate.py          # §8.1, §8.5, §8.6 – synthetic, no network
python scripts/corpus_eval.py      # §4.1 – fetches the Commons corpus on first run
python scripts/evaluate.py <root> eval.json      # §8.2, §8.3
python scripts/evaluate_text.py <dir> text.json  # §8.7
python scripts/evaluate_dibco.py <dibco> d.json  # §8.4
python scripts/build_report.py     # regenerate the PDF of this document
```

Datasets used

Corpus	What it gives	Licence
Wikimedia Commons (189 QR + 56 barcode + 139 text)	real photographs	CC0 / PD / CC BY / CC BY-SA, per-file attribution in <code>inkmap/data/commons_qr.json</code>
DIBCO / H-DIBCO 2009–2019	per-pixel ink ground truth, documents	no licence stated; cite the per-year competition papers. Used for internal validation only, not redistributed

Datasets deliberately **not** used: ICDAR RRC (the site requires free registration; static paths that appear to bypass it were not used), TextSeg and BTS (access-walled). For future text work, Hi-SAM’s HierText stroke masks (11,639 images, per-pixel, Apache-2.0 masks over CC BY-SA 4.0 images) are the best openly available per-pixel text ink ground truth.

References

1. [ISO/IEC 18004 — QR Code bar code symbology specification](#)
2. [ISO/IEC 15415 — Bar code print quality test specification, two-dimensional symbols](#)
3. [ISO/IEC 15416 — Bar code print quality test specification, linear symbols](#)
4. [ISO 12647-2:2013 — Graphic technology, process control \(§3.1 area coverage\)](#)
5. [IEC 61966-2-1 — sRGB colour space](#)
6. [DIBCO / H-DIBCO document image binarization competitions](#)
7. [zxing-cpp](#) — Apache-2.0 decoder used for location and as the round-trip oracle

8. [Evangelidis & Psarakis, *Parametric image alignment using enhanced correlation coefficient maximization \(ECC\)*](#)
9. [Sauvola & Pietikäinen, *Adaptive document image binarization*](#)
10. [Keshava & Mustard, *Spectral unmixing*](#) — the mixed-pixel framing
11. [Soret, Bacharach & Buvat, *Partial-volume effect in PET tumor imaging*](#) — the contamination result in its original field
12. [Zamberletti et al., *ArTe-Lab 1D barcode datasets*](#) — CC BY 3.0, region-level ground truth
13. [BoofCV QR code benchmark](#) — CC BY 4.0, quad-polygon ground truth