

# Converting Hand-Drawn Lines into Vector Segments

Hand-drawn circuit sketches are typically noisy, wobbly and composed of thick strokes rather than perfect straight lines. A robust approach is to first **preprocess and thin** the image before fitting segments. For example, Kelly and Cole (2024) begin by binarizing the scanned schematic (e.g. via Otsu's method) and applying a Zhang-Suen thinning to produce a **one-pixel-wide skeleton** of every wire <sup>1</sup>. This cleans up most of the stroke "bulk" so that each hand-drawn line becomes a 1-pixel curve. The skeleton can be analyzed to find end-points and junctions (pixels with only one neighbor) which often correspond to wire endpoints or symbol leads <sup>2</sup> <sup>3</sup>. For example, the figure below (from Kelly & Cole) shows a typical hand-sketched circuit and the result of thinning it to a skeleton with PPHT-detected lines:

*Figure: Example hand-drawn circuit (adapted from Kelly & Cole 2024) after thresholding and thinning. In this 1-pixel skeleton, wire segments can be detected by fitting straight lines (e.g. using a Probabilistic Hough transform) <sup>1</sup>. Components (circles, symbols) have been identified separately.*

The basic steps in a sketch-to-netlist pipeline are thus: 1. **Preprocessing:** Smooth or denoise the image (e.g. Gaussian blur) and apply global or adaptive thresholding to get a clean black-and-white image <sup>1</sup>. Remove tiny specks or fill small gaps with morphology if needed.

2. **Skeletonization:** Thin the binary lines down to one-pixel width. Methods like Zhang-Suen or more advanced *unbiased* thinning can remove stroke width while preserving topology <sup>3</sup> <sup>1</sup>. The result is a topological skeleton of the sketch.

3. **Line extraction:** Fit straight segments to parts of the skeleton. One approach is to run a **probabilistic Hough transform** (PPHT) on the skeleton to find maximal straight runs of pixels <sup>1</sup>. (Kelly & Cole's pipeline, for instance, uses PPHT with length thresholds and local checks to label true wire segments <sup>1</sup>.) Alternatively, iterative RANSAC or LSD (Line Segment Detector) can be applied: repeatedly find a line that fits many skeleton pixels, remove its inliers, and iterate to find all dominant lines. These methods convert wiggly paths into candidate line segments.

4. **Post-processing segments:** Merge or prune the fitted segments. Because hand-drawn lines aren't perfectly straight, one wire may be split into several detected pieces. You can merge colinear or nearly-colinear adjacent segments (for example by checking if their endpoints are close and directionally aligned). Pruning removes tiny spurs or false detections (e.g. require a minimum length or check that neighboring pixels support it). In Donati et al.'s framework, for example, variants of pruning, merging and edge-linking are used to clean the one-pixel paths <sup>3</sup>.

5. **Vectorization:** Convert each clean line segment into a precise geometric object. A simple approach is to treat each fitted segment as a straight line (with two endpoints). More advanced vectorization (e.g. for curved strokes) fits Bézier or spline curves. Donati *et al.* (2019) use a modified vectorization to output few control points for each stroke <sup>3</sup>. Similarly, modern methods like "simplicity" use the medial axis or polyvector fields to produce high-quality curve vectors <sup>4</sup> <sup>5</sup>. In any case, the goal is to represent each wire as a parameterized line or polyline.

6. **Junction and netlist extraction:** Once lines are vectorized, find their intersection or touching points to determine circuit nodes. For axis-aligned wires (like many schematics), one can simply take every

horizontal/vertical segment endpoint and mark a node where they intersect <sup>2</sup>. More generally, build a graph whose nodes are the line endpoints and crossings, and whose edges are the line segments between them. Each symbol (resistor, capacitor, etc.) touches one or more node points. From this graph you can generate a netlist by listing each component and which nodes it connects. For example, Kelly & Cole (2024) form horizontal and vertical lists of wires, then label a node whenever a horizontal and vertical segment overlap at endpoints <sup>2</sup>.

In summary, a practical pipeline is: **binarize→thin→detect line segments→clean/merge→vectorize**. This combination of skeletonization and robust line fitting is widely recommended. Egiazarian & Voynov (2020) note that many vectorization systems start by extracting a curve network via skeleton initialization and thinning <sup>4</sup>, then simplify it to fit primitives. Donati *et al.* (2019) likewise emphasize that a novel thinning step produces clean 1-pixel lines from scribbles <sup>3</sup>. Recent work (e.g. Magne & Sorkine 2025) even employs graph-based medial-axis analyses to ensure line order and connectivity when vectorizing single-stroke drawings <sup>5</sup>.

Apart from classical image methods, machine learning approaches are emerging. For example, Netlistify (Huang *et al.*, 2025) uses deep networks to recognize both components and connectivity in schematics <sup>6</sup>. These end-to-end tools can handle complex cases by training on many examples. However, they still rely on the same underlying idea of extracting wire segments to build a netlist graph.

**Key references and tools:** Donati *et al.* (2019) provide a complete sketch vectorization framework with robust line extraction and thinning algorithms <sup>3</sup>. Egiazarian & Voynov (ECCV 2020) survey vectorization for technical drawings, noting skeleton-based curve-net extraction <sup>4</sup>. Magne & Sorkine-Hornung (2025) focus on single-stroke drawings using medial-axis graphs <sup>5</sup>. Kelly & Cole (APL Machine Learning 2024) demonstrate a full pipeline for schematics: threshold → skeleton → PPHT line detection → junction finding <sup>1</sup> <sup>2</sup>. Finally, Netlistify (2025) tackles conversion to netlists via deep learning <sup>6</sup>, illustrating that sophisticated connectivity extraction is now feasible.

**Sources:** State-of-the-art methods are drawn from digital sketch/vectorization literature <sup>3</sup> <sup>4</sup> <sup>5</sup> and recent schematic-recognition studies <sup>1</sup> <sup>2</sup> <sup>6</sup>, as cited above. These works address noisy hand-drawn lines via thinning, fitted line extraction, and graph-based vectorization.

---

<sup>1</sup> <sup>2</sup> (PDF) Digitizing images of electrical-circuit schematics

[https://www.researchgate.net/publication/377742261\\_Digitizing\\_images\\_of\\_electrical-circuit\\_schematics](https://www.researchgate.net/publication/377742261_Digitizing_images_of_electrical-circuit_schematics)

<sup>3</sup> (PDF) A complete hand-drawn sketch vectorization framework

[https://www.researchgate.net/publication/323257011\\_A\\_complete\\_hand-drawn\\_sketch\\_vectorization\\_framework](https://www.researchgate.net/publication/323257011_A_complete_hand-drawn_sketch_vectorization_framework)

<sup>4</sup> [ecva.net](https://www.ecva.net)

[https://www.ecva.net/papers/eccv\\_2020/papers\\_ECCV/papers/123580579.pdf](https://www.ecva.net/papers/eccv_2020/papers_ECCV/papers/123580579.pdf)

<sup>5</sup> (PDF) Single-Line Drawing Vectorization

[https://www.researchgate.net/publication/396238058\\_Single-Line\\_Drawing\\_Vectorization](https://www.researchgate.net/publication/396238058_Single-Line_Drawing_Vectorization)

<sup>6</sup> [research.nvidia.com](https://research.nvidia.com)

[https://research.nvidia.com/labs/electronic-design-automation/papers/netlistify\\_mlcad25.pdf](https://research.nvidia.com/labs/electronic-design-automation/papers/netlistify_mlcad25.pdf)