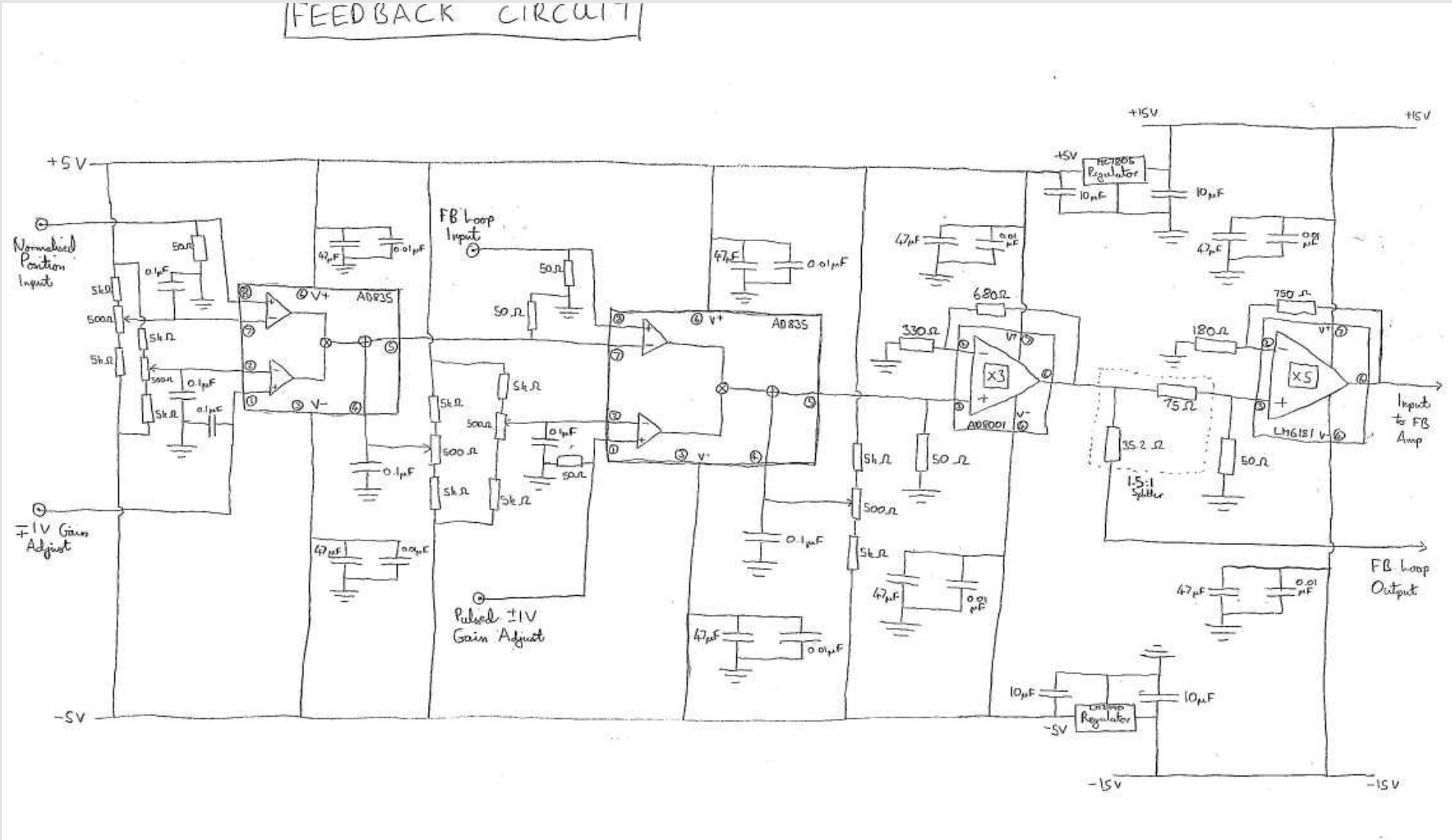
Interactive Recognition of Hand-drawn Circuit diagrams

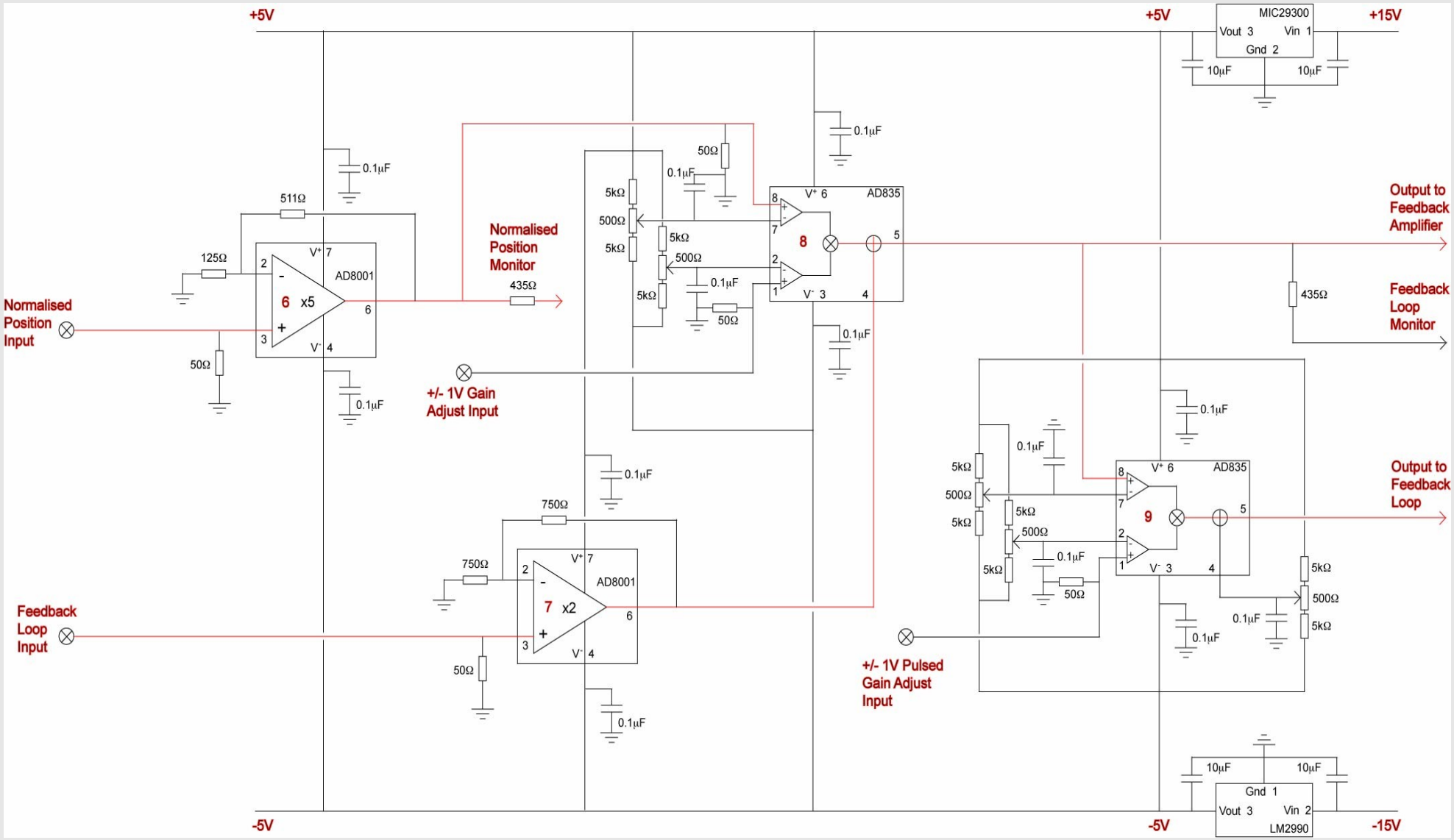
CAD in 30 seconds

Initial steps in the design of an electronic circuit...

Step 1: Napkin design

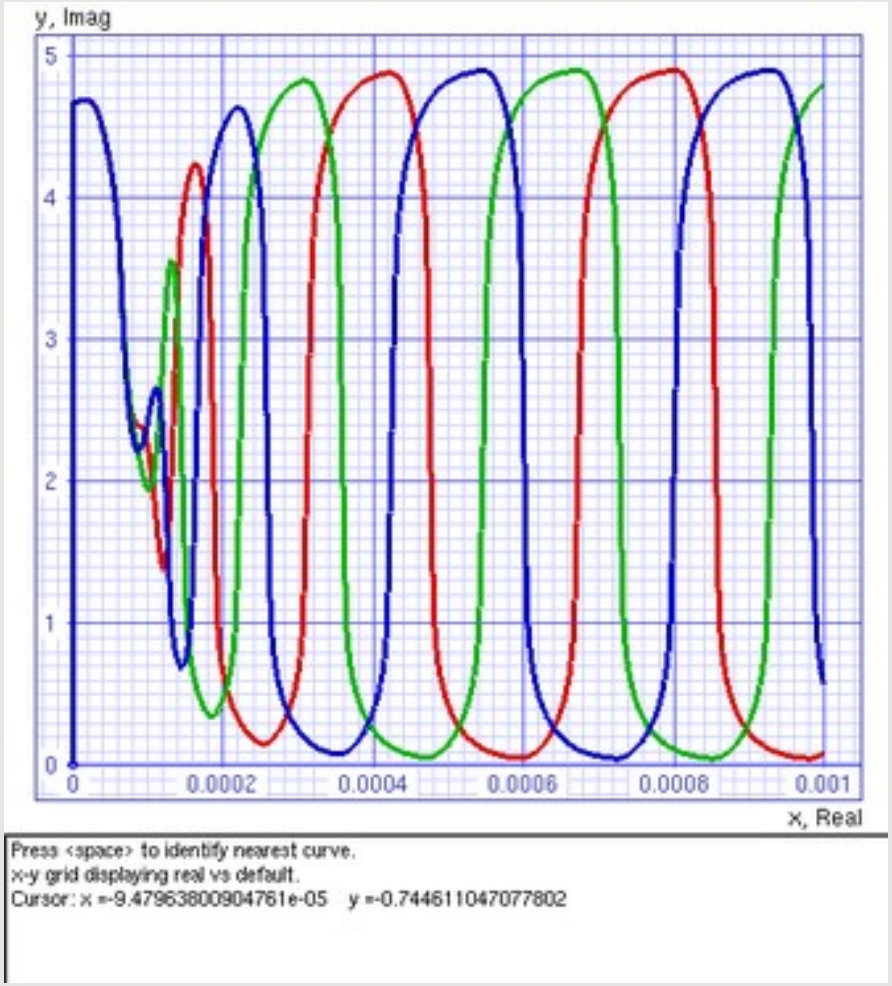


Step 2: Capture Circuit



Step 3: SPICE & Simulate

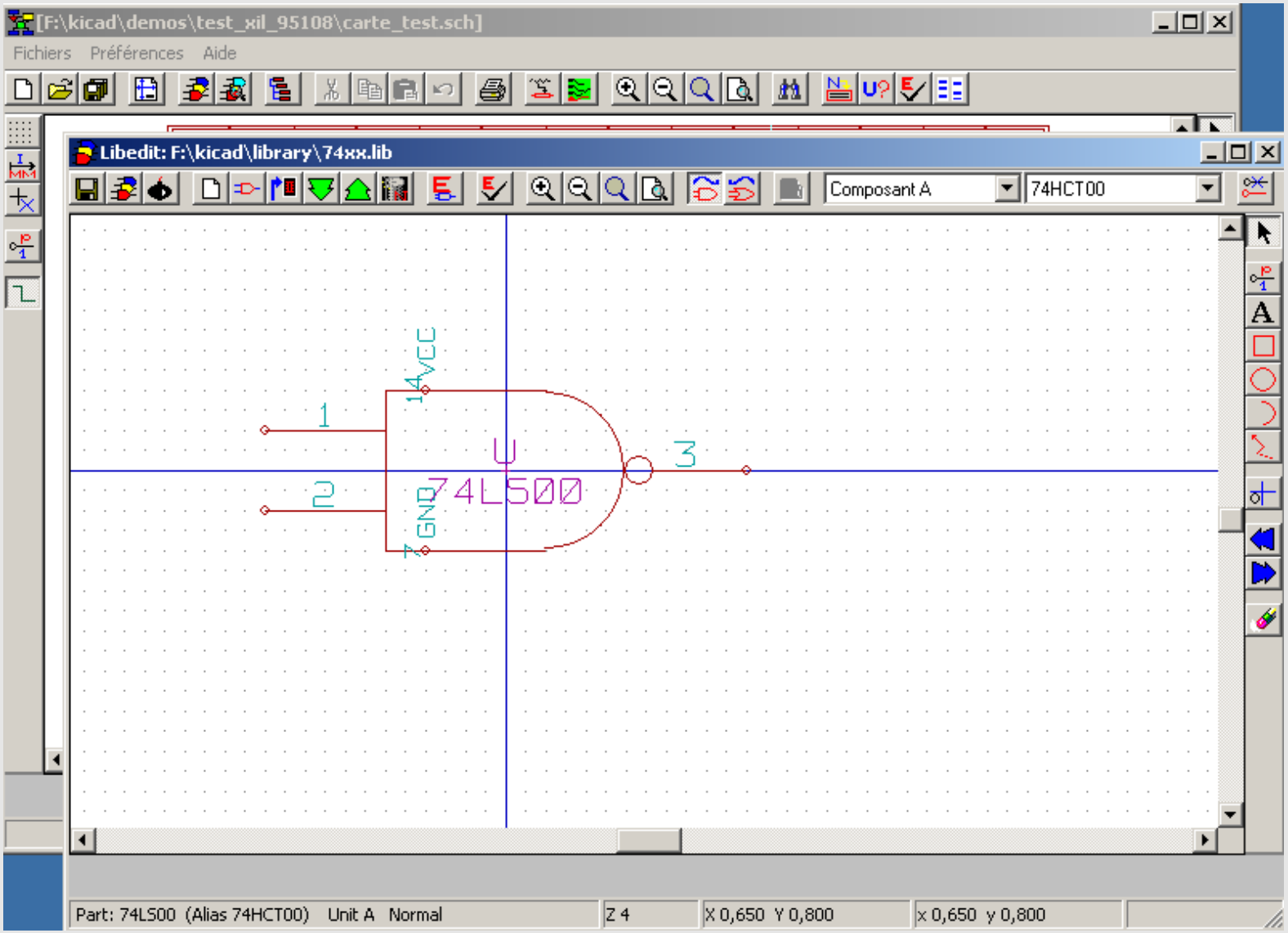
- VS 1 0 AC 1 PWL(0US 0V 0.01US 1V 100US 1V)
- VCC 10 0 DC +5V
- VEE 11 0 DC -5V
- R1 0 2 1
- R2 2 3 1
- XOP 1 2 3 10 11 OPAMP3
- RL 3 0 100K
- .SUBCKT OPAMP3 1 2 81 101 102
- Q1 5 1 7 NPN
- Q2 6 2 8 NPN
- RC1 101 5 151.7
- RC2 101 6 151.7
- RE1 7 4 100
- RE2 8 4 100
- I1 4 102 0.001
- GV 100 15 6 5 0.001
- RV 15 100 200K
- DZ1 15 16 DZENER
- DZ2 100 16 DZENER
- G1 100 10 15 100 0.0005
- RP1 10 100 1MEG
- CP1 10 100 79.6PF
- EOUT 80 100 10 100 1
- RO 80 81 100
- RREF1 101 103 100K
- RREF2 103 102 100K
- EREF 100 0 103 0 1
- R100 100 0 1MEG
- .MODEL NPN NPN(BF=50000)
- .MODEL DZENER D(BV=5.7V IS=1E-14 IBV=1E-3)
- .ENDS
- .TRAN 0.001US 0.2US
- …



Why napkin design?

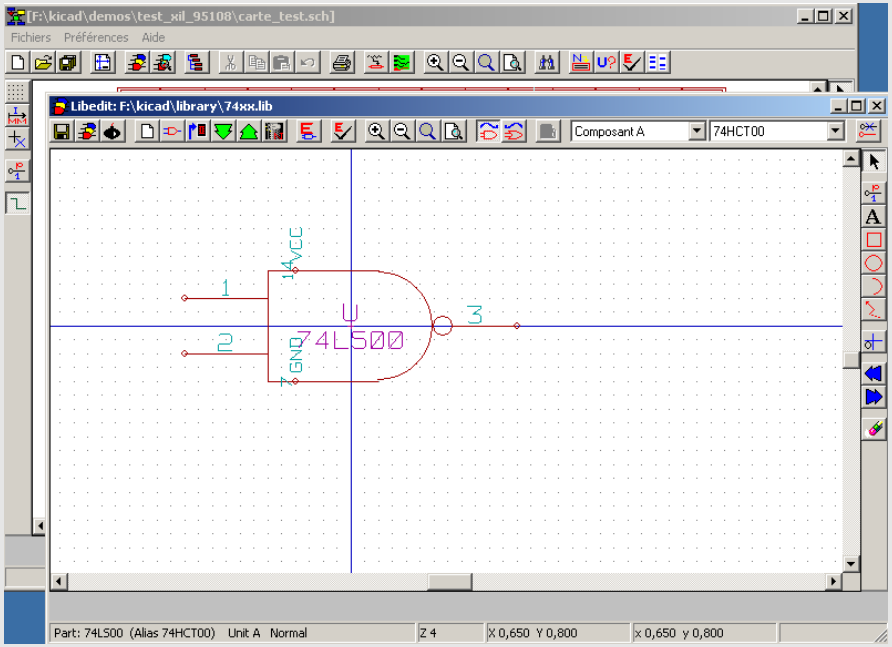
- Why initial design on paper?
- Why not design directly in CAD environment?

Why napkin design?



Why napkin design?

- **CAD software is non-intuitive**
- Click-and-drop from library
- Rotating, moving and connecting together components is awkward an error-prone



Why napkin design?

- **Pen-and-paper is mobile**
- Fast prototyping!

What is a TabletPC?

- Fully functional laptop
- Operate with stylus or digital pen instead of a keyboard or mouse
- Handwriting recognition
- Windows XP Tablet PC Edition

Why TabletPC?

- Mobility
- Pen-based – natural feel
- CAD tools (SPICE)

The Vision

- **Sketch** circuit directly onto Tablet PC
- Automatically interpreted as **circuit diagram**
- **SPICE** generated from circuit diagram

=> *“circuit_sketch”*

“circuit_sketch”

Requirements

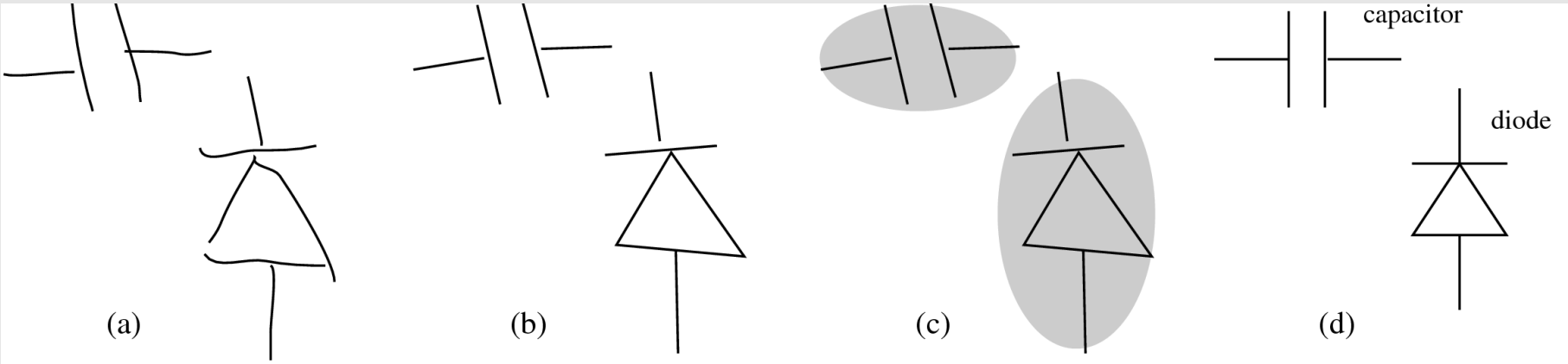
- interactive – fast and responsive
- customizable – only one training example per component

Pure Python

- Rapid development
- Numarray libraries for speed

“circuit_sketch”

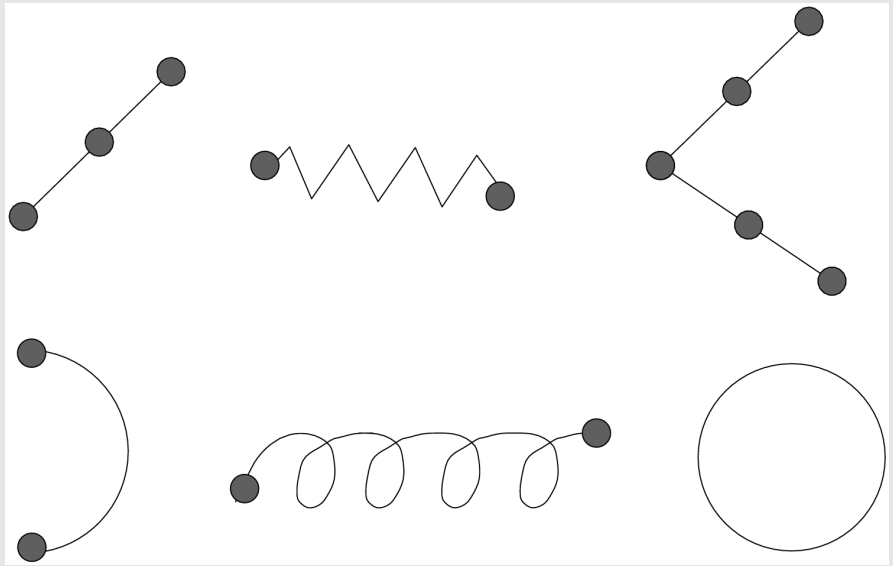
(a) Capture strokes
(b) Identify primitives
(c) Cluster into symbols
(d) Recognize symbols



Primitive Identification

Component Symbols built from primitives:

- Line
- Jagged
- Corner
- Arc
- Spiral
- Circle



Primitive identification

- Least squares estimate
- Fluid sketching
- Primitive is updated as drawn

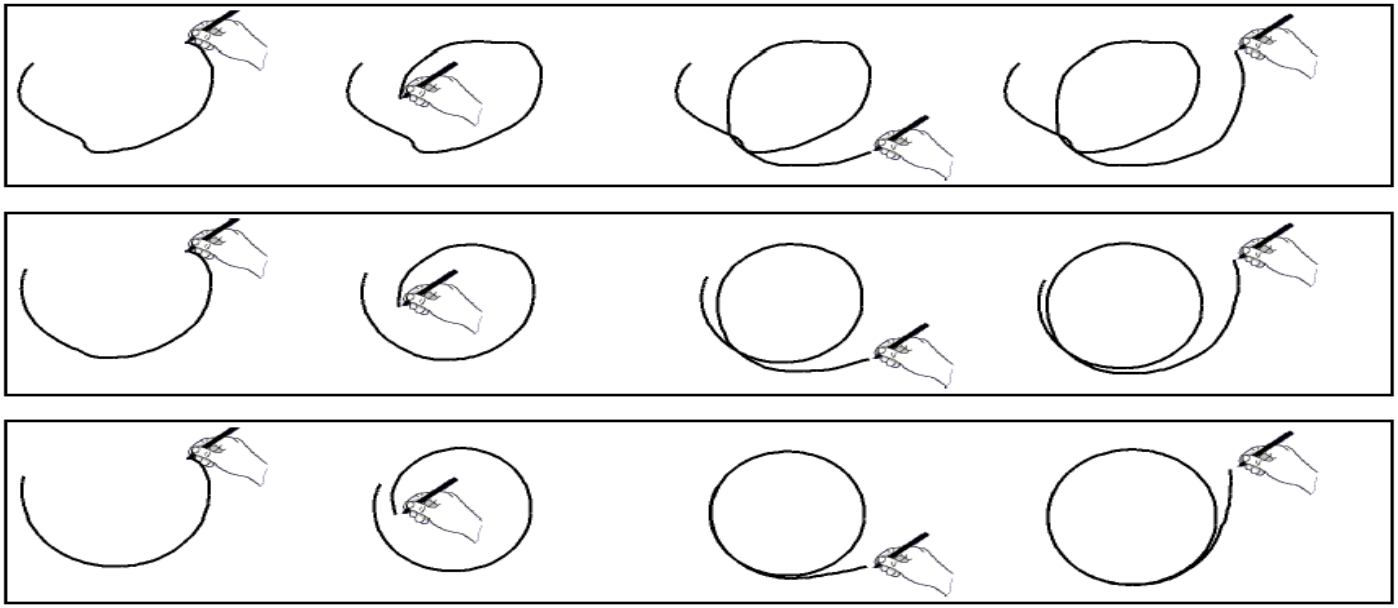
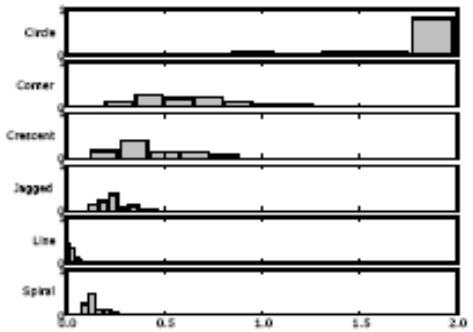
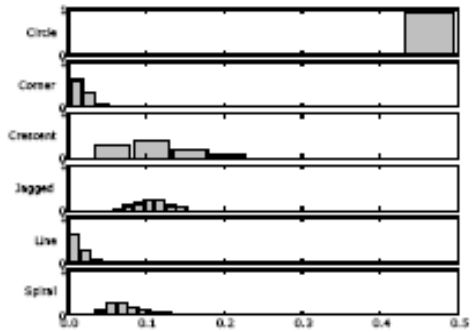


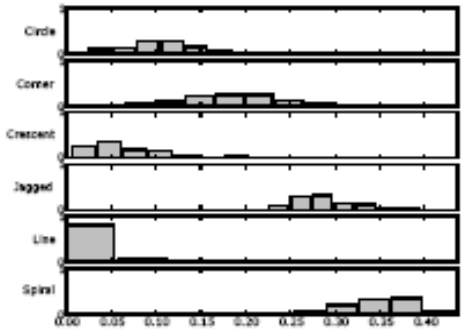
Figure 4: *Top row: a roughly circular pattern is drawn by the user. This data was captured as a user interacted with our prototype system. Middle row: with fluid sketching enabled, the very same trajectory continuously morphs toward the least-squares circle. Bottom row: when the viscosity is decreased, the morphing reaches the current optimal shape more quickly.*



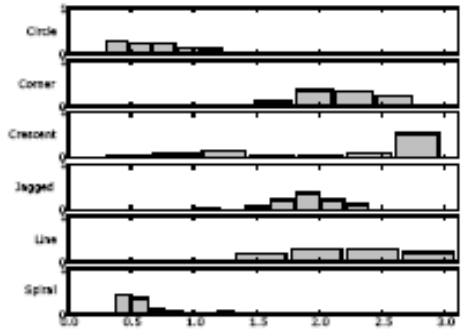
(a) deviation of points from line estimate



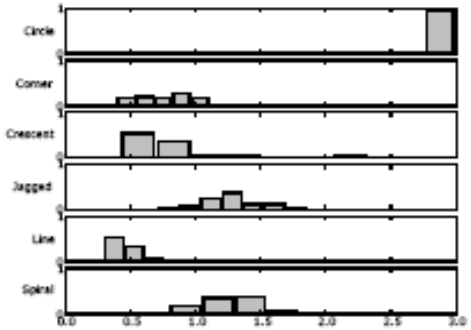
(b) deviation of points from corner estimate



(c) distance from radius of estimated circle



(d) angular changes between points

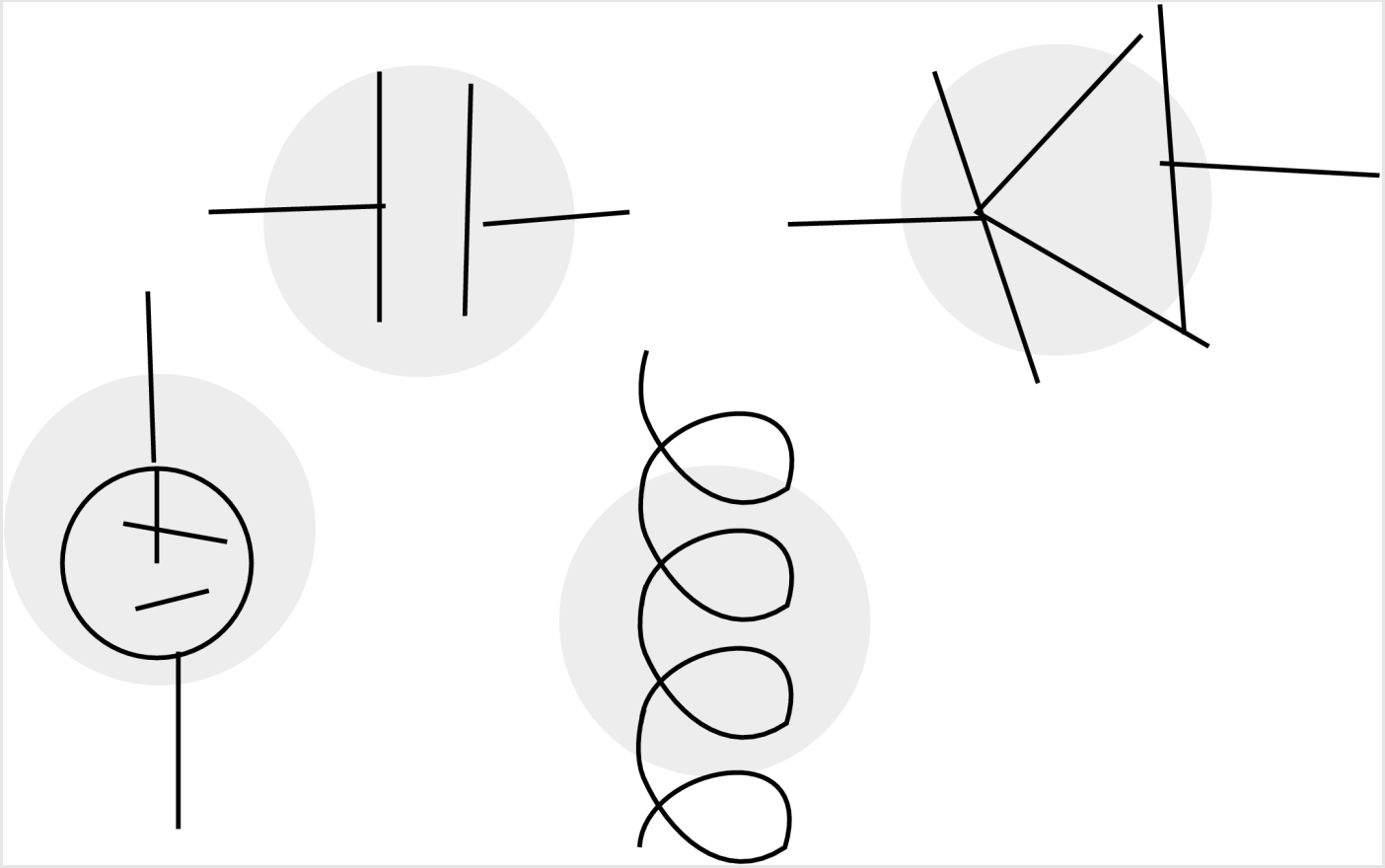


(e) stroke curvature length

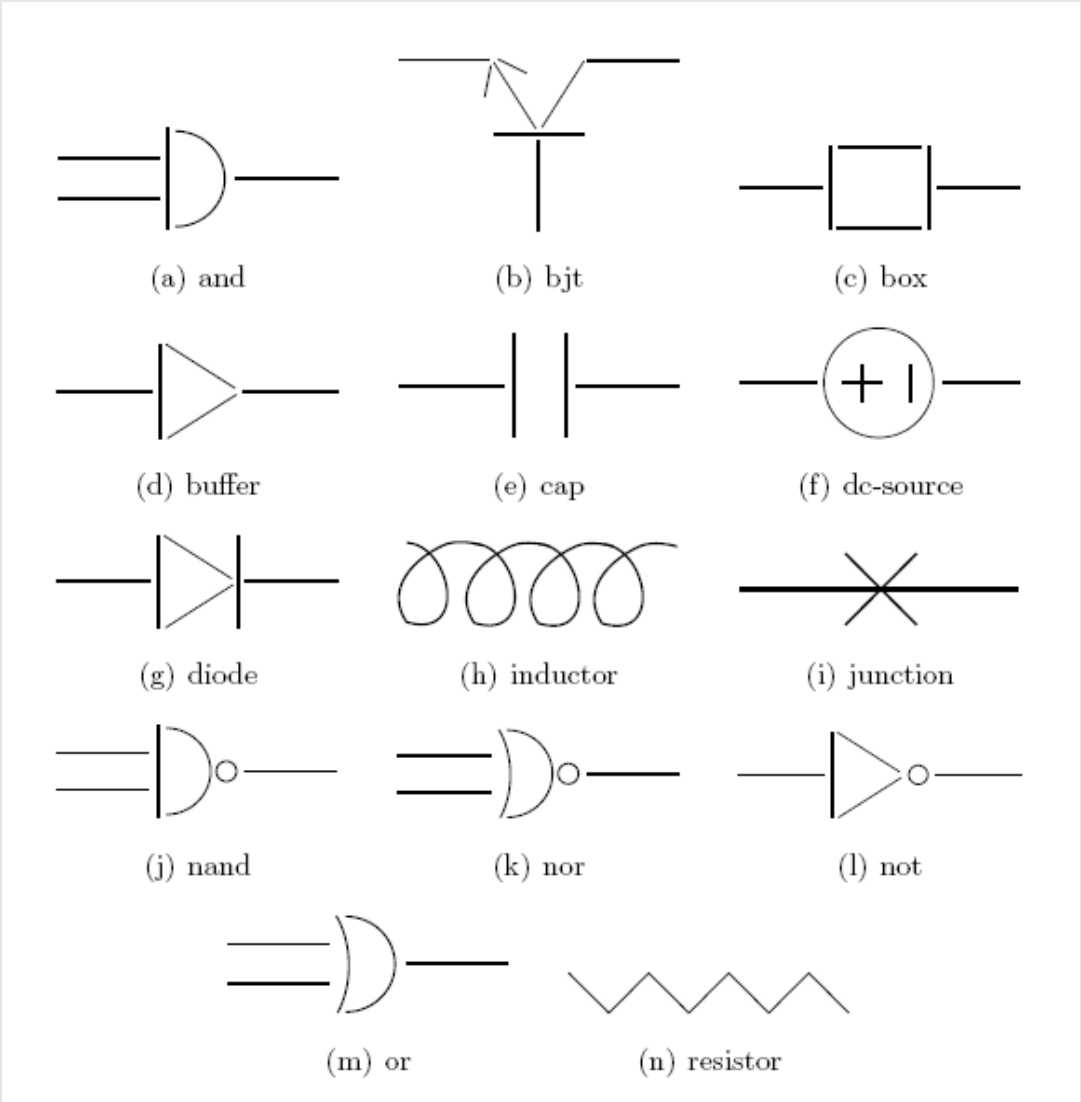
Primitives clustered into symbols

k-means

- <x, y, time>



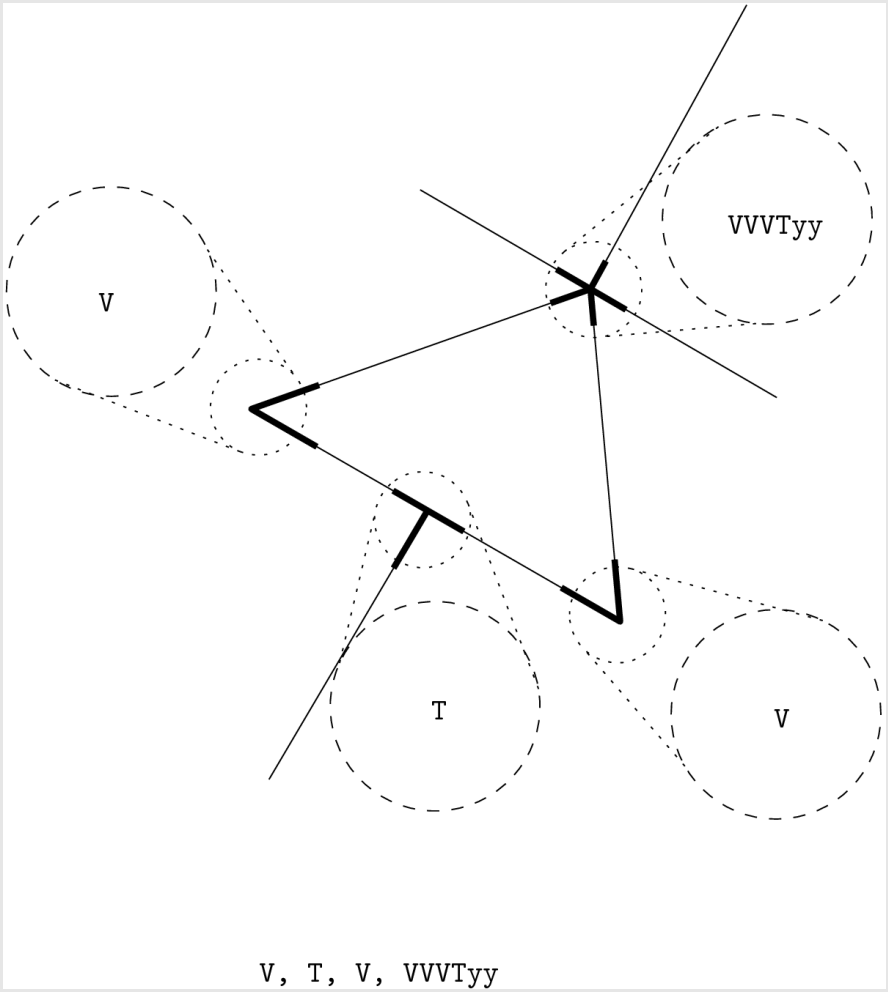
Symbol definitions



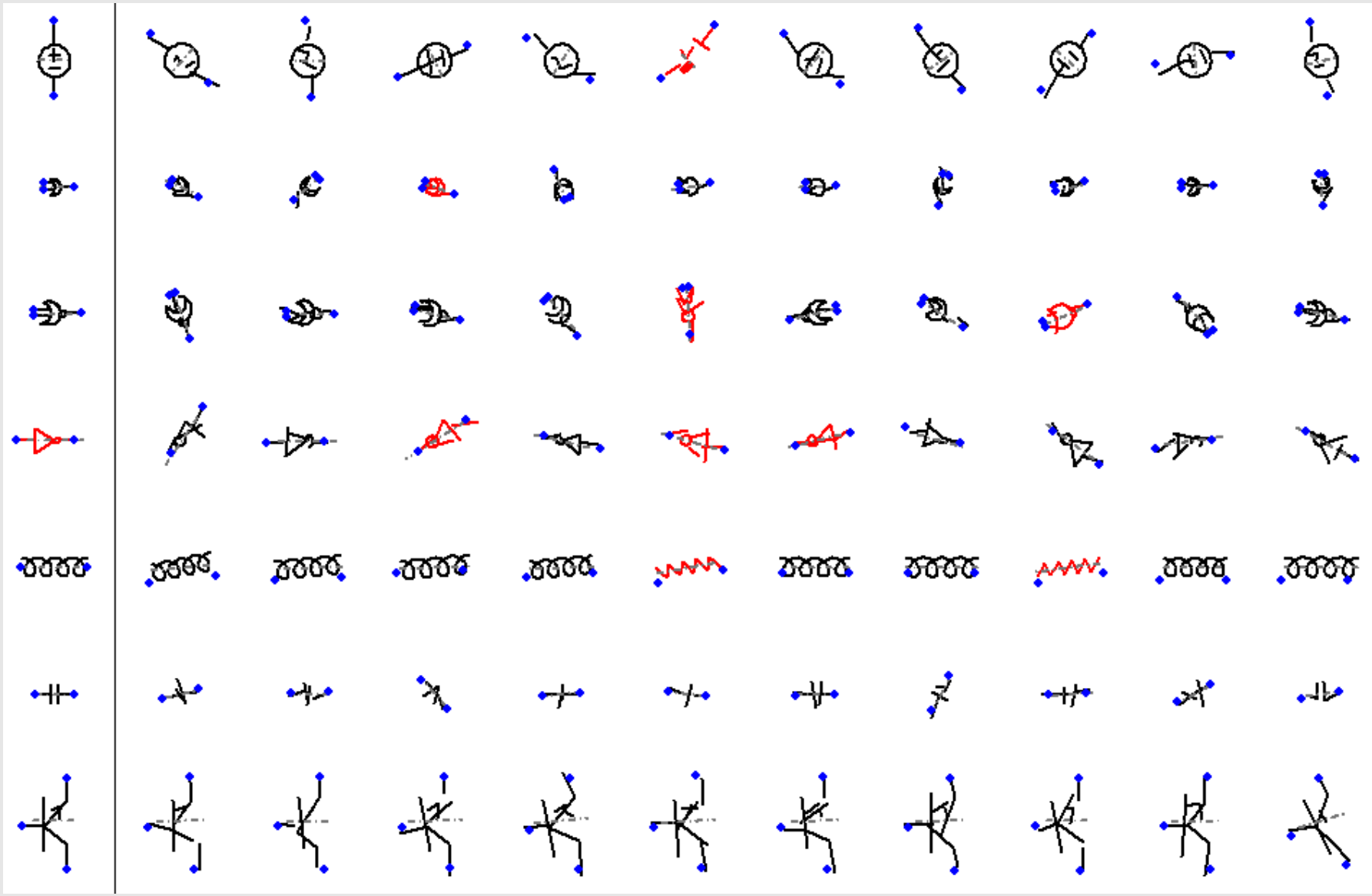
Intersection features

- 6 Lines
- 0 Arcs
- 0 Circles
- 5 V-intersections
- 2 T-intersections
- 2 y-intersections
- ...

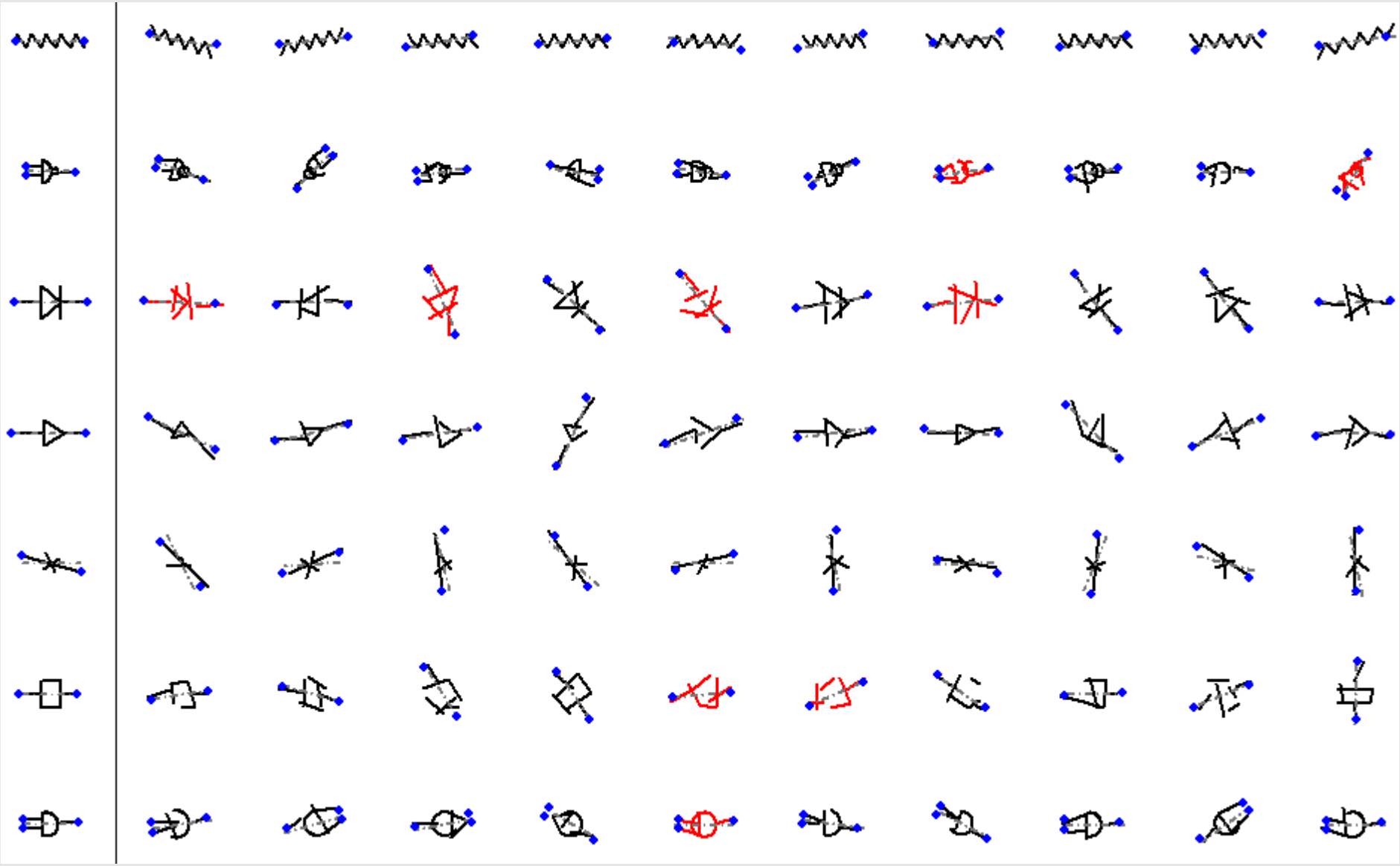
<6, 0, 0, 5, 2, 2, ...>



Features matched against examples...

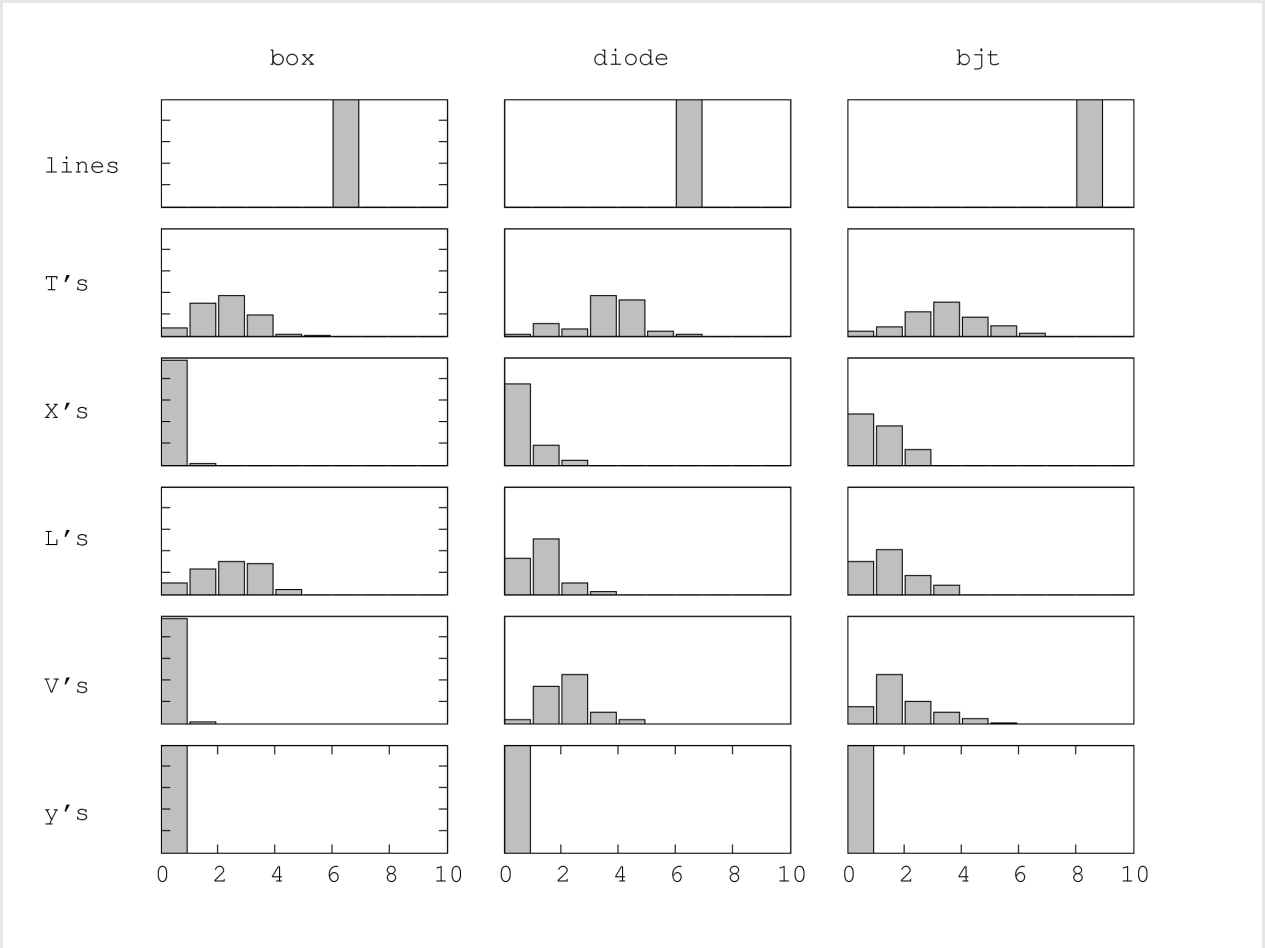


...generated from a single symbol definition

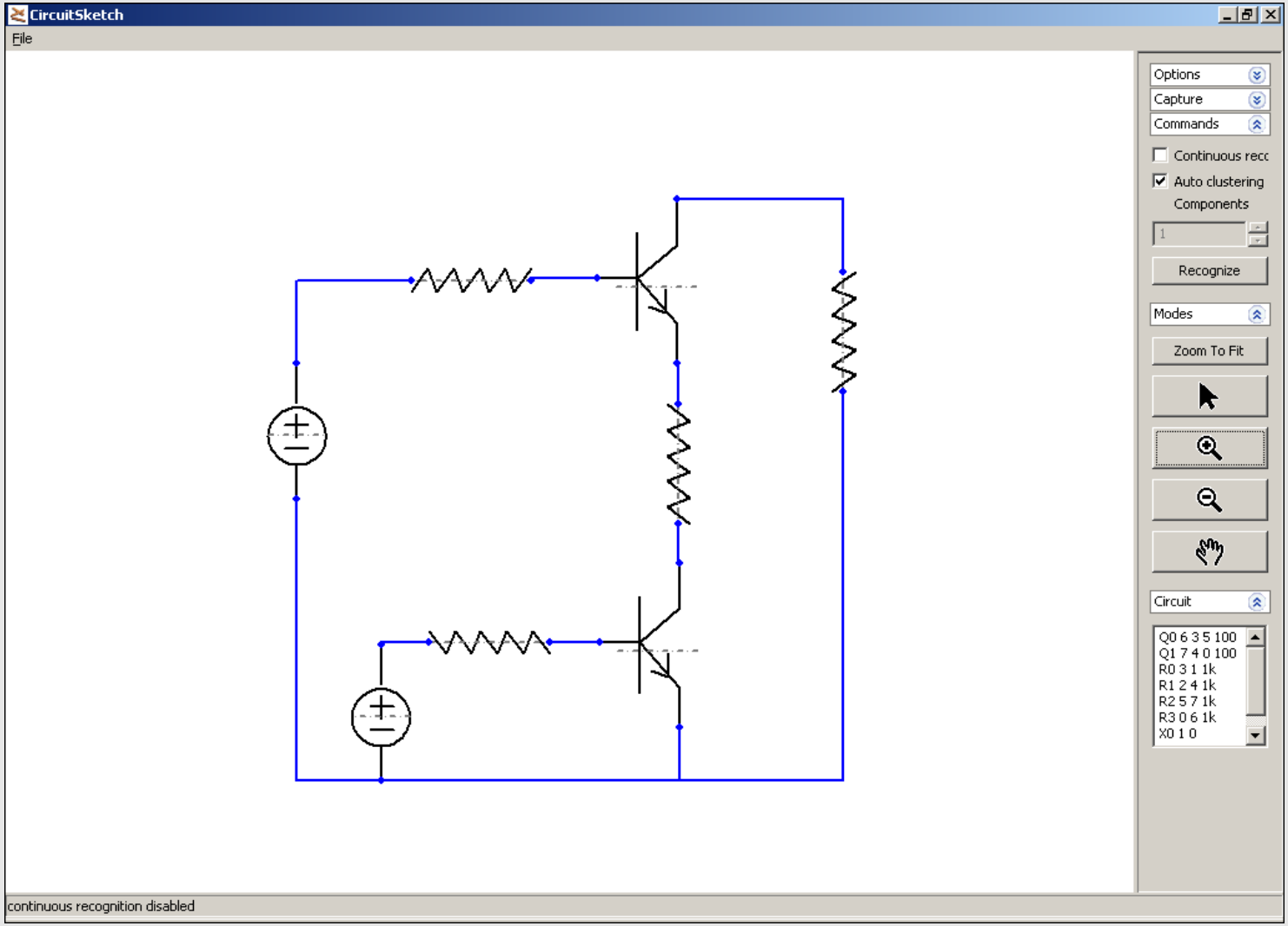


Symbol Recognition

- discrete Bayesian classifier (i.e. histograms)



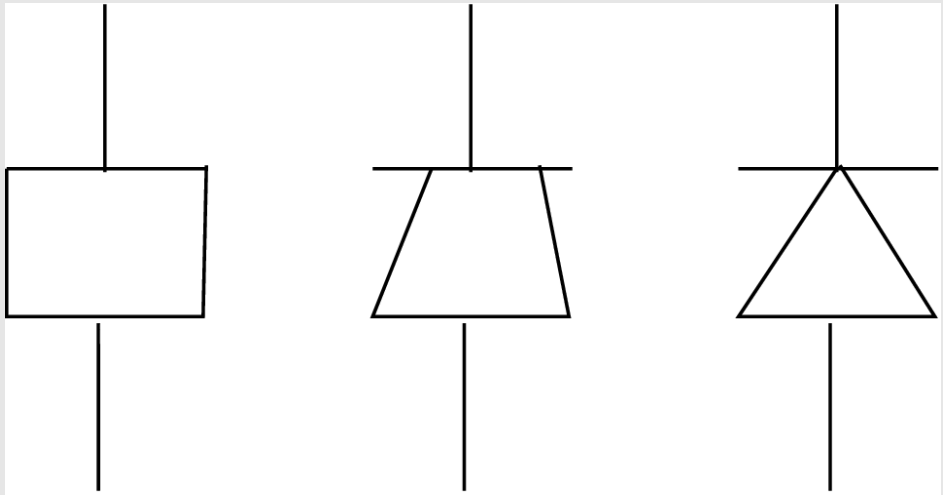
Finally...
Symbols connected together

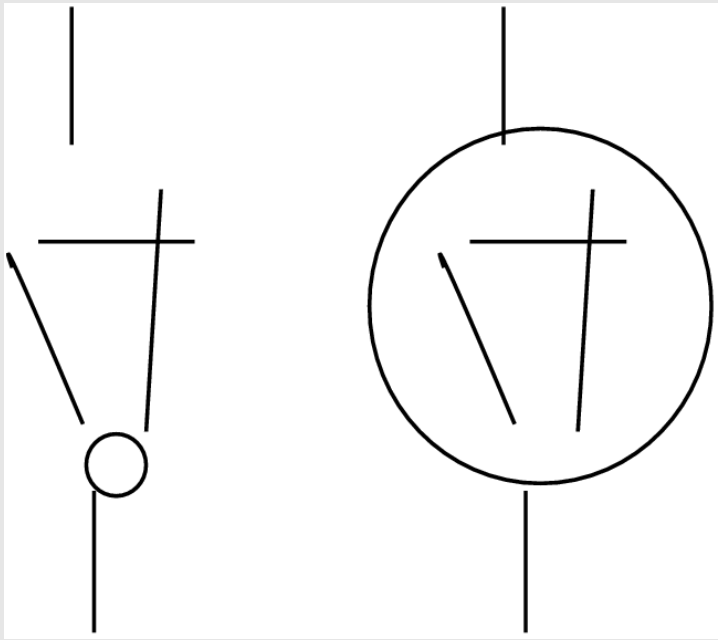


Limitations

"If a machine is expected to be infallible,
it cannot also be intelligent."
- Alan Turing,
20 February 1947

Limitations





Conclusion

It works

- Except for large circuits

Improvements

- Clustering (larger circuits)
- Polygons (triangles, rectangles...)

Future

- Hybrid system (sketch narrows down search)