In the following pseudocode, POINT is a structure that expresses \( x, y \), and strokeId properties. strokeId is the stroke index a point belongs to (1, 2, ...). It's also filled by counting pen down/up events. POINTS is a list of points and TEMPLATES is a list of POINTS with gesture class data.

### Recognizer main function

Match points against a set of templates by employing the Nearest-Neighbor classification rule.

- Returns a normalized score in \([0,1]\) with 1 denoting perfect match.

#### \$\text{Recognizer} \ (\text{Points \ points, Templates \ templates})$

1. \( n \leftarrow 32 \) // number of points
2. \( \text{Normalize}(\text{points}, n) \)
3. \( \text{score} \leftarrow \infty \)
4. for each \( \text{template} \) in \( \text{templates} \) do
5. \( \text{Normalize}(\text{template}, n) \)
6. \( d \leftarrow \text{GREEDY-CLOUD-MATCH}(\text{points}, \text{template}, n) \)
7. if \( \text{score} > d \) then
8. \( \text{score} \leftarrow d \)
9. result \( \leftarrow \text{template} \)
10. return \( (\text{result}, \text{score}) \)

### Greedy-CLOUD-MATCH (Points points, Points template, int n)

1. \( \epsilon \leftarrow 0.01 \)
2. \( \text{step} \leftarrow [1.01] \)
3. \( \text{min} \leftarrow \infty \)
4. for \( i = 0 \) to \text{step} \ do
5. \( d_1 \leftarrow \text{CLOUD-DISTANCE}(\text{points}, \text{template}, n, i) \)
6. \( d_2 \leftarrow \text{CLOUD-DISTANCE}(\text{template}, \text{points}, n, i) \)
7. \( \text{min} \leftarrow \min(\text{min}, d_1, d_2) \)
8. \( \text{return} \text{min} \)

### CLOUD-DISTANCE (Points points, Points template, int n, int start)

1. \( \text{matched} \leftarrow \text{new \ boolean}[n] \)
2. \( \text{sum} \leftarrow 0 \)
3. \( i \leftarrow \text{start} \) // start matching with points
4. \( \text{do} \)
5. \( \text{min} \leftarrow \infty \)
6. for each \( j \) such that \( \text{not matched}[j] \) do
7. \( d \leftarrow \text{EUCLIDEAN-DISTANCE}(\text{points}_i, \text{tmpl}_j) \)
8. if \( d < \text{min} \) then
9. \( \text{min} \leftarrow d \)
10. \( \text{index} \leftarrow j \)
11. \( \text{matched}[\text{index}] \leftarrow \text{true} \)
12. \( \text{weight} \leftarrow 1 + ((\text{start} + n) \ \text{MOD} \ n)/n \)
13. \( \text{sum} \leftarrow \text{sum} + \text{weight} \cdot \text{min} \)
14. \( i \leftarrow (i + 1) \ \text{MOD} \ n \)
15. until \( i = \text{start} \) // all points are processed
16. \( \text{return} \text{sum} \)

The following pseudocode addresses gesture preprocessing (or normalization) which includes resampling, scaling with shape preservation, and translation to origin. The code is similar to \$1$ and SN recognizers\(^1,2\) and we repeat it here for completeness. We highlight minor changes.

#### \$\text{Normalize} \ (\text{Points \ points, int n})$

1. \( \text{points} \leftarrow \text{RESAMPLE}(\text{points}, n) \)
2. \( \text{SCALE}(\text{points}) \)
3. \( \text{TRANSLATE-TO-ORIGIN}(\text{points}, n) \)

### Points resampling

Resample a points path into \( n \) evenly spaced points. We use \( n = 32 \).

#### \$\text{RESAMPLE} \ (\text{Points \ points, int n})$

1. \( \text{nP} \leftarrow \text{PATH-LENGTH}(\text{points}) / (n - 1) \)
2. \( D \leftarrow 0 \)
3. \( \text{newPoints} \leftarrow \text{points} \)
4. for each \( \text{point} \) in \( \text{points} \) such that \( i > 0 \) do
5. if \( p_i . \text{strokeId} \ == \ p_{i-1} . \text{strokeId} \) then
6. \( d \leftarrow \text{EUCLIDEAN-DISTANCE}(p_{i-1}, p_{i}) \)
7. if \( (D + d) \geq I \) then
8. \( q \times \leftarrow p_{i-1} . x - (I - D)/d \cdot (p_{i} . x - p_{i-1} . x) \)
9. \( q \cdot y \leftarrow p_{i-1} . y + (I - D)/d \cdot (p_{i} . y - p_{i-1} . y) \)
10. \( \text{append}(\text{newPoints}, q) \)
11. \( \text{INSERT}(\text{points}, q) \)
12. \( q \) will be the next \( p_i \)
13. \( D \leftarrow D + d \)
14. else \( D = D + d \)
15. return \( \text{newPoints} \)

#### \$\text{PATH-LENGTH} \ (\text{Points \ points})$

1. \( D \leftarrow 0 \)
2. for each \( \text{point} \) in \( \text{points} \) such that \( i > 0 \) do
3. if \( p_i . \text{strokeId} \ == \ p_{i-1} . \text{strokeId} \) then
4. \( D \leftarrow D + \text{EUCLIDEAN-DISTANCE}(p_{i-1}, p_{i}) \)
5. return \( D \)

### Points rescale

Rescale points with shape preservation so that the resulting bounding box will be \([0,1] \times [0,1] \).

#### \$\text{SCALE} \ (\text{Points \ points})$

1. \( x_{\min} \leftarrow \infty, x_{\max} \leftarrow 0 \)
2. \( y_{\min} \leftarrow \infty, y_{\max} \leftarrow 0 \)
3. \( x_{\min} \leftarrow \text{MIN}(x_{\min}, p.x) \)
4. \( y_{\min} \leftarrow \text{MIN}(y_{\min}, p.y) \)
5. \( x_{\max} \leftarrow \text{MAX}(x_{\max}, p.x) \)
6. \( y_{\max} \leftarrow \text{MAX}(y_{\max}, p.y) \)
7. \( \text{scale} \leftarrow \text{MAX}(x_{\max} - x_{\min}, y_{\max} - y_{\min}) \)
8. for each \( p \) in \( \text{points} \) do
9. \( p \leftarrow ((p.x - x_{\min}) / \text{scale}, (p.y - y_{\min}) / \text{scale}) \)
10. \( \text{p.strokeId} \)

### Points translate

Translate points to the origin \((0,0)\).

#### \$\text{TRANSLATE-TO-ORIGIN} \ (\text{Points \ points, int n})$

1. \( c \leftarrow (0, 0) \) // will contain centroid
2. for each \( p \) in \( \text{points} \) do
3. \( c \leftarrow (c.x + p.x \cdot c.y + p.y) \)
4. \( c \leftarrow (c.x/n, c.y/n) \)
5. for each \( p \) in \( \text{points} \) do
6. \( p \leftarrow (p.x - c.x, p.y - c.y) \)
7. \( \text{p.strokeId} \)

---

\(^1\)http://depts.washington.edu/aimgroup/proj/dollar/index.html
\(^2\)http://depts.washington.edu/aimgroup/proj/dollar/index.html

This pseudocode is modified slightly from that which appears in the original ACM ICMI 2012 publication by Vatavu, Anthony, and Wobbrock (http://dx.doi.org/10.1145/2388676.2388732) to better highlight the use of the strokeId property and to provide a normalized matching score. It also contains more comments to assist the implementation. This algorithm’s logic remains unchanged.