Step 1. Resample a points path into n evenly spaced points. We use n=64. For gestures serving as templates, Steps 1-3 should be carried out once on the raw input points. For candidates, Steps 1-4 should be used just after the candidate is articulated.

Resample(points, n)
1 I ← PATH-LENGTH(points) / (n - 1)
2 D ← 0
3 newPoints ← points0
4 foreach point pi for i ≥ 1 in points do
5 d ← DISTANCE(pi-1, pi)
6 if (D + d) ≥ I then
7 qi ← pi-1 + ((I - D) / d) × (pi - pi-1)
8 else qi ← pi-1 + ((I - D) / d) × (pi - pi-1)
9 APPEND(newPoints, qi)
10 return newPoints

Path-Length(A)
1 d ← 0
2 for i from 1 to |A| step 1 do
3 d ← d + DISTANCE(Ai, Ai)
4 return d

Step 2. Find and save the indicative angle ω from the points’ centroid to first point. Then rotate by -ω to set this angle to 0°.

Indicative-Angle(points)
1 c ← CENTROID(points) // computes (x̄, ȳ)
2 return ATAN(cx - points0x, cy - points0y) // for -π ≤ ω ≤ π

Rotate-By(points, ω)
1 c ← CENTROID(points)
2 foreach point p in points do
3 qi ← (p - cx) × COS ω - (p - cy) × SIN ω + cx
4 qj ← (p - cx) × SIN ω + (p - cy) × COS ω + cy
5 APPEND(newPoints, qi)
6 return newPoints

Step 3. Scale points so that the resulting bounding box will be of size’ size. We use size’=250. Then translate points to the origin k=(0,0). Bounding-Box returns a rectangle defined by (minx, miny), (maxx, maxy).

Scale-To(points, size)
1 B ← Bounding-Box(points)
2 foreach point p in points do
3 qi ← px × size / Bwidth
4 qj ← py × size / Bheight
5 APPEND(newPoints, qi)
6 return newPoints

Translate-To(points, k)
1 c ← CENTROID(points)
2 foreach point p in points do
3 qi ← px + kx - cx
4 qj ← py + ky - cy
5 APPEND(newPoints, qi)
6 return newPoints

Step 4. Match points against a set of templates. The size variable on line 7 of RECOGNIZE refers to the size passed to Scale-To in Step 3. The symbol $ϕ equals [√(1 + √5)]. We use $θ=±45° and $θ0=2° on line 3 of RECOGNIZE. Due to using RESAMPLE, we can assume that A and B in Path-Distance contain the same number of points, i.e., |A|=|B|.

Recognize(points, templates)
1 b ← +∞
2 foreach template T in templates do
3 d ← DISTANCE-AT-BEST-ANGLE(points, T, -θ, +θ)
4 if d < b then
5 b ← d
6 T ← T
7 return (T, score)

Distance-At-Best-Angle(points, T, θa, θb, θ0)
1 x1 ← ϕθ0 + (1 - ϕθb)
2 f1 ← DISTANCE-AT-Angle(points, T, x1)
3 x2 ← (1 - ϕθa) + ϕθb
4 f2 ← DISTANCE-AT-Angle(points, T, x2)
5 while |θa - θb| > 2θ0 do
6 if f1 < f2 then
7 θa ← x2
8 x2 ← x1
9 f2 ← f1
10 x1 ← ϕθa + (1 - ϕθb)
11 f1 ← DISTANCE-AT-Angle(points, T, x1)
12 else
13 θb ← x1
14 x1 ← x2
15 f1 ← f2
16 x2 ← (1 - ϕθa) + ϕθb
17 f2 ← DISTANCE-AT-Angle(points, T, x2)
18 return MIN(f1, f2)

Distance-At-Angle(points, T, θ)
1 newPoints ← Rotate-By(points, θ)
2 d ← Path-Distance(newPoints, Tpoints)
3 return d

Path-Distance(A, B)
1 d ← 0
2 for i from 0 to |A| step 1 do
3 d ← d + Distance(Ai, Bi)
4 return d / |A|

This pseudocode is modified slightly from that which appears in the original ACM UIST 2007 publication by Wobbrock, Wilson and Li to be parallel to the more recent $SV$ multistroke recognizer. This algorithm’s logic remains unchanged.