Step 1. Take a multistroke gesture *strokes* and generate unistroke permutations. For gestures serving as templates, Step 1, which uses Steps 3-6, should be carried out once on the input points. For candidates, Steps 2-7 should be applied to the input points. For constants we use N=96, *size*=250, ∂=.30, O=(0,0), and I=12.

```
GENERATE-UNISTROKE-PERMUTATIONS(strokes)
1   for i from 0 to |strokes| do order_i ← i
2   HEAP-PERMUTE(|strokes|, order, out orders)
3   M ← MAKE-UNISTROKES(strokes, orders)
4   foreach unistroke U in M do
5      U_points ← RESAMPLE(U_points, N)   // step 3
6      ω ← INDICATIVE-ANGLE(U_points)   // step 4
7      U_points ← ROTATE-BY(U_points, −ω)
8      U_points ← SCALE-DIM-TO(U_points, size, ∂)   // step 5
9      U_points ← CHECK-RESTORE-ORIENTATION(U_points, +ω)
10     U_points ← TRANSLATE-TO(U_points, O)
11     U_vector ← CALC-START-UNIT-VECTOR(U_points, I)   // step 6
HEAP-PERMUTE(n, order, out orders)
1   if n = 1 then APPEND(orders, order)
2   else
3      for i from 0 to n do
4         HEAP-PERMUTE(n-1, order, out orders)
5         if IS-ODD(n) then SWAP(order_0, order_n-1)
6         else SWAP(order_i, order_n-1)
MAKE-UNISTROKES(strokes, orders)
1   foreach order R in orders do
2      for b from 0 to 2^|R| do
3         for i from 0 to |R| do
4            if BIT-AT(b, i) = 1 then   // b's bit at index i
5               APPEND(unistroke, REVERSE(strokes_R_i))
6            else APPEND(unistroke, strokes_R_i)
7         APPEND(unistrokes, unistroke)
8   return unistrokes
```

Step 2. Combine candidate *strokes* into one unistroke *points* path.

```
COMBINE-STROKES(strokes)
1   for i from 0 to |strokes| do
2      for j from 0 to |strokes_i| do
3         APPEND(points, strokes_i_j)   // append each point
4   return points
```

Step 3. Resample a *points* path into n evenly spaced points.

```
RESAMPLE(points, n)
1   I ← PATH-LENGTH(points) / (n − 1)
2   D ← 0
3   newPoints ← points_0
4   foreach point p_i for i ≥ 1 in points do
5      d ← DISTANCE(p_i-1, p_i)
6      if (D + d) ≥ I then
7         q_x ← p_i-1_x + ((I − D) / d) × (p_i_x − p_i-1_x)
8         q_y ← p_i-1_y + ((I − D) / d) × (p_i_y − p_i-1_y)
9         APPEND(newPoints, q)
10        INSERT(points, i, q)   // q will be the next p_i
11        D ← 0
12     else D ← D + d
13  return newPoints
PATH-LENGTH(A)
1   d ← 0
2   for i from 1 to |A| step 1 do
3      d ← d + DISTANCE(A_i-1, A_i)
4   return d
```

Step 4. Find and save the indicative angle ω from the *points*' centroid to first point. Then rotate by $-\omega$ to set this angle to 0°.

```
INDICATIVE-ANGLE(points)
1   c ← CENTROID(points)   // computes (x̄, ȳ)
2   return ATAN(c_y − points_0_y, c_x − points_0_x)   // for -π ≤ ω ≤ π
ROTATE-BY(points, ω)
1   c ← CENTROID(points)
2   foreach point p in points do
3      q_x ← (p_x − c_x) COS ω − (p_y − c_y) SIN ω + c_x
4      q_y ← (p_x − c_x) SIN ω + (p_y − c_y) COS ω + c_y
5      APPEND(newPoints, q)
6   return newPoints
```

Step 5. Scale dimensionally-sensitive based on threshold ∂=.30. Next, *if* using bounded rotation invariance, restore drawn orientation by rotating $+\omega$. Then translate to the origin O=(0,0).

```
SCALE-DIM-TO(points, size, ∂)
1   B ← BOUNDING-BOX(points)
2   foreach point p in points do
3      if MIN(B_width / B_height, B_height / B_width) ≤ ∂ then   // uniform
4         q_x ← p_x × size / MAX(B_width, B_height)
5         q_y ← p_y × size / MAX(B_width, B_height)
6      else   // non-uniform
7         q_x ← p_x × size / B_width
8         q_y ← p_y × size / B_height
9      APPEND(newPoints, q)
10  return newPoints
CHECK-RESTORE-ORIENTATION(points, ω)
1   if using bounded rotation invariance then
2      points ← ROTATE-BY(points, ω)
3   return points
TRANSLATE-TO(points, k)
1   c ← CENTROID(points)
2   foreach point p in points do
3      q_x ← p_x + k_x − c_x
4      q_y ← p_y + k_y − c_y
5      APPEND(newPoints, q)
6   return newPoints
```

Step 6. Calculate the start unit vector v for *points* using index I=12.

```
CALC-START-UNIT-VECTOR(points, I)
1   q_x ← points_I_x − points_0_x
2   q_y ← points_I_y − points_0_y
3   v_x ← q_x / √(q_x^2 + q_y^2)
4   v_y ← q_y / √(q_x^2 + q_y^2)
5   return v
```

Step 7. Match candidate *points* having start unit vector v, processed from the raw *strokes* in Step 2, where now S = |*strokes*|, against unistroke permutations U within each multistroke M. We use Φ = 30° for the start angle similarity threshold and *size*=250. The symbol φ equals ½(-1 + √5). We pass θ=±45° and θ_Δ=2°.

```
RECOGNIZE(points, v, S, multistrokes)
1   b ← +∞
2   foreach multistroke M in multistrokes do
3      if S = |M_strokes| then   // optional: require same # strokes
4         foreach unistroke U in M do
5            if ANGLE-BETWEEN-VECTORS(v, U_vector) ≤ Φ then
6               d ← DISTANCE-AT-BEST-ANGLE(points, U, −θ, θ, θ_Δ)
7               if d < b then b ← d, M′ ← M
8   score ← 1 − b / [½√(size^2 + size^2)]
9   return ⟨M′, score⟩
ANGLE-BETWEEN-VECTORS(A, B)
1   return ACOS(A_x × B_x + A_y × B_y)
```

```
DISTANCE-AT-BEST-ANGLE(points, T, θa, θb, θΔ)
 1  x1 ← φθa + (1 − φ)θb
 2  f1 ← DISTANCE-AT-ANGLE(points, T, x1)
 3  x2 ← (1 − φ)θa + φθb
 4  f2 ← DISTANCE-AT-ANGLE(points, T, x2)
 5  while |θb − θa| > θΔ do
 6     if f1 < f2 then
 7        θb ← x2
 8        x2 ← x1
 9        f2 ← f1
10        x1 ← φθa + (1 − φ)θb
11        f1 ← DISTANCE-AT-ANGLE(points, T, x1)
12     else
13        θa ← x1
14        x1 ← x2
15        f1 ← f2
16        x2 ← (1 − φ)θa + 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, T_points)
 3  return d
PATH-DISTANCE(A, B)
 1  d ← 0
 2  for i from 0 to |A| step 1 do
 3     d ← d + DISTANCE(A_i, B_i)
 4  return d / |A|
```