$N-Protractor Pseudocode

Lisa Anthony[†] and Jacob O. Wobbrock[‡]

[†]University of Maryland Baltimore County, [‡]The Information School, DUB Group, University of Washington

The Protractor-enhanced $N pseudocode is presented here to aid implementation. In addition, the $N webpage[1] includes open-source JavaScript and C# implementations. Operations that have not changed since the original $N are noted; for those details, see [1]. For results of the $N-Protractor evaluations, see [2].

Step 1. Take a multistroke gesture *strokes* and generate unistroke permutations of it. For gestures serving as templates, Step 1, which uses Steps 3-7, should be carried out once on the input points. For candidates, Steps 2-8 should be applied to the input points. For constants we use N=96, *size*=250, ∂=.30, O=(0,0), and I=12. HEAP-PERMUTE and MAKE-UNISTROKES are unchanged from Step 1 in [1].

```
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_start ← CALC-START-UNIT-VECTOR(U_points, I)   // step 6
12     U_vector ← VECTORIZE(U_points)   // step 7
```

Step 2. Combine candidate *strokes* into one unistroke *points* path. COMBINE-STROKES remains unchanged from Step 2 in [1].

Step 3. Resample a *points* path into n evenly spaced points. RESAMPLE remains unchanged from Step 1 in [4].

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 and ROTATE-BY remain unchanged from Step 4 in [1].

Step 5. Scale in a dimensionally-sensitive fashion 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, TRANSLATE-TO and CHECK-RESTORE-ORIENTATION remain unchanged from Step 5 in [1].

Step 6. Calculate the start unit vector v_{start} for *points* using index I=12. CALC-START-UNIT-VECTOR remains unchanged from Step 6 in [1].

Step 7. Create the vector version *vector* of *points*. This step has been changed for $N-Protractor from Protractor's original formulation [3] to remove calls to RESAMPLE and TRANSLATE-TO, since $N-Protractor already performs these operations in Steps 3 and 5.

```
VECTORIZE(points)
 1  c ← 1
 2  s ← 0
 3  if using bounded rotation invariance then
 4     θ_first ← ATAN(points_0y, points_0x)
 5     θ_base ← (π/4) × FLOOR[(θ_first + π/8)/(π/4)]
 6     c ← COS(θ_base − θ_first)
 7     s ← SIN(θ_base − θ_first)
 8  sum ← 0
 9  foreach point p in points do
10     x_new ← p_x × c − p_y × s
11     y_new ← p_y × c + p_x × s
12     APPEND2(vector, x_new, y_new)
13     sum ← sum + x_new^2 + y_new^2
14  magnitude ← √sum
15  foreach value v in vector do
16     v ← v / magnitude
17  return vector
```

Step 8. Match vector version *vector* of candidate *points* having start unit-vector v_{start}, processed from the raw *strokes* in Steps 2-7 prior to calling RECOGNIZE, where now S = |*strokes*|, against unistroke permutations U within each multistroke M. We use Φ = 30° for the start angle similarity threshold. OPTIMAL-COSINE-DISTANCE is the matching method from Protractor [3].

```
RECOGNIZE(points, vector, v_start, 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_start, U_start) ≤ Φ then
 6              d ← OPTIMAL-COSINE-DISTANCE(vector, U_vector)
 7              if d < b then b ← d, M′ ← M
 8  score ← 1 − b
 9  return ⟨M′, score⟩
ANGLE-BETWEEN-VECTORS(v_1, v_2)
 1  return ACOS(v_1x × v_2x + v_1y × v_2y)
OPTIMAL-COSINE-DISTANCE(v_1, v_2)
 1  a ← 0
 2  b ← 0
 3  for i from 0 to |v_1| step 2 do
 4     a ← a + v_1i × v_2i + v_1(i+1) × v_2(i+1)
 5     b ← b + v_1i × v_2(i+1) − v_1(i+1) × v_2i
 6  ψ ← ATAN(b / a)
 7  return ACOS(a × COS ψ + b × SIN ψ)
```

[1] http://depts.washington.edu/aimgroup/proj/dollar/ndollar.html

REFERENCES

[1] Anthony, L. and Wobbrock, J.O. (2010). A lightweight multistroke recognizer for user interface prototypes. Proceedings of Graphics Interface (GI '10). Ottawa, Ontario (May 31-June 2, 2010). Toronto, Ontario: Canadian Information Processing Society, pp. 245-252.

[2] Anthony, L. and Wobbrock, J.O. (2012). $N-Protractor: A fast and accurate multistroke recognizer. Proceedings of Graphics Interface (GI '12). Toronto, Ontario (May 28-30, 2012). Toronto, Ontario: Canadian Information Processing Society, pp. 117-120.

[3] Li, Y. (2010). Protractor: A fast and accurate gesture recognizer. Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI '10). Atlanta, Georgia (April 10-15, 2010). New York: ACM Press, pp. 2169-2172.

[4] Wobbrock, J.O., Wilson, A.D. and Li, Y. (2007). Gestures without libraries, toolkits or training: A $1 recognizer for user interface prototypes. Proceedings of the ACM Symposium on User Interface Software and Technology (UIST '07). Newport, Rhode Island (October 7-10, 2007). New York: ACM Press, pp. 159-168.