**PROTRACTOR**\(^1\) **PSEUDO CODE**\(^2\)

Step 1: Resample the points of a gesture into \(n\) evenly spaced points. Protractor uses the same resampling method as the \(1\) recognizer\(^3\) does, although Protractor only needs \(n = 16\) points to perform optimally. The pseudo code of this step is borrowed from the \(1\) recognizer.

**RESAMPLE** \((\text{points}, n)\)

1. \(I \leftarrow \text{PATH-LENGTH}(\text{points}) \div (n - 1)\)
2. \(D \leftarrow 0\)
3. \(\text{newPoints} \leftarrow \text{points}_0\)
4. **foreach** point \(p_i\) for \(i \geq 1\) in \(\text{points} \) **do**
   5. \(d \leftarrow \text{DISTANCE}(p_{i-1}, p_i)\)
   6. **if** \((D + d) \geq I\) **then**
      7. \(q_x \leftarrow p_{i-1} + ((I - D) \div d) \times (p_{i} - p_{i-1})\)
      8. \(q_y \leftarrow p_{i-1} + ((I - D) \div d) \times (p_{i} - p_{i-1})\)
      9. **APPEND** (\(\text{newPoints}, q\))
     10. **INSERT** \((\text{points}, i, q)\)  // \(q\) will be the next \(p_i\)
   11. \(D \leftarrow 0\)
   12. **else** \(D \leftarrow D + d\)
      13. **return** \(\text{newPoints}\)

**PATH-LENGTH** \((A)\)

1. \(\text{distance} \leftarrow 0\)
2. **for** \(i\) from 1 to \(|A|\) **step** 1 **do**
3. \(\text{distance} \leftarrow \text{distance} + \text{DISTANCE}(A_{i-1}, A_i)\)
4. **return** \(\text{distance}\)

Step 2: Generate a vector representation for the gesture. The procedure takes two parameters: points are resampled points from Step 1, and \(\text{oSensitive}\) specifies whether the gesture should be treated orientation sensitive or invariant. The procedure first translates the gesture so that its centroid is the origin, and then rotates the gesture to align its indicative angle with a base orientation. **VECTORIZE** returns a normalized vector with a length of \(2n\).

**VECTORIZE** \((\text{points}, \text{oSensitive})\)

1. \(\text{centroid} \leftarrow \text{CENTROID}(\text{points})\)
2. \(\text{points} \leftarrow \text{TRANSLATE}(\text{points}, \text{centroid})\)
3. \(\text{indicativeAngle} \leftarrow \text{ATAN}(\text{points}_0, \text{points}_0)\)
4. **if** \(\text{oSensitive}\) **then**
   5. \(\text{baseOrientation} \leftarrow (\pi / 4) \cdot \text{FLOOR}((\text{indicativeAngle} + \pi / 8) / (\pi / 4))\)
   6. \(\text{delta} \leftarrow \text{baseOrientation} - \text{indicativeAngle}\)
5. **else** \(\text{delta} \leftarrow -\text{indicativeAngle}\)
6. \(\text{sum} \leftarrow 0\)
7. **foreach** point \((x, y)\) in \(\text{points}\) **do**
   8. \(\text{newX} \leftarrow x \text{COS}(\text{delta}) - y \text{SIN}(\text{delta})\)
   9. \(\text{newY} \leftarrow y \text{COS}(\text{delta}) + x \text{SIN}(\text{delta})\)
10. **APPEND** \((\text{vector}, \text{newX}, \text{newY})\)
11. \(\text{sum} \leftarrow \text{sum} + \text{newX} \times \text{newX} + \text{newY} \times \text{newY}\)
15 magnitude ← SQRT (sum)
16 foreach e in vector do
17 e ← e / magnitude
18 return vector

Step 3: Match the vector of an unknown gesture against a set of templates. OPTIMAL-
COSINE-DISTANCE provides a closed-form solution to find the minimum cosine distance
between the vectors of a template and the unknown gesture by only rotating the template
once.

RECOGNIZE (vector, templates)
1 maxScore ← 0
2 foreach template in templates do
3 distance ← OPTIMAL-COSINE-DISTANCE (templatevector, vector)
4 score ← 1 / distance
5 if score > maxScore then
6 maxScore ← score
7 match ← template_name
8 return (match, score)

OPTIMAL-COSINE-DISTANCE (vector, vector’)
1 a = 0
2 b = 0
3 for i from 1 to vector Step 2 do
4 a ← a + vector_i × vector’_i + vector_i+1 × vector’_i+1
5 b ← b + vector_i × vector’_i+1 – vector_i+1 × vector’_i
6 angle ← ATAN (b/a)
7 return ACOS (a × COS (angle) + b × SIN (angle))

1 Yang Li, Protractor: a fast and accurate gesture recognizer, CHI 2010: ACM Conference

2 This pseudocode is formatted based on the $1 pseudocode written by Jacob O.
Wobbrock. See the $1 recognizer at http://depts.washington.edu/aimgroup/proj/dollar/.
Step 1 is the same as the preprocessing step in the $1 recognizer. The key differences of
Protractor lie in Step 2 and 3, in which Protractor 1) uses Cosine distances instead of
Euclidean distances, 2) supports orientations of gestures, and 3) employs a closed-form
one-step rotation to acquire a minimum Cosine distance between gestures.

3 Jacob Wobbrock, Andy Wilson, Yang Li, Gestures without libraries, toolkits or
Training: a $1.00 Recognizer for User Interface Prototypes, UIST 2007: ACM Symposium
on User Interface Software and Technology. p.159-168.