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 (points, n)
    $I \leftarrow$ PATH-LENGTH (points) / $(n - 1)$
    $D \leftarrow 0$
    $newPoints \leftarrow points_0$
    foreach point $p_i$ for $i \geq 1$ in points do
        $d \leftarrow$ DISTANCE$(p_{i-1}, p_i)$
        if $(D + d) \geq I$ then
            $q_x \leftarrow p_{i-1_x} + ((I - D) / d) \times (p_{i_x} - p_{i-1_x})$
            $q_y \leftarrow p_{i-1_y} + ((I - D) / d) \times (p_{i_y} - p_{i-1_y})$
            APPEND (newPoints, q)
            INSERT (points, i, q)   // q will be the next $p_i$
            $D \leftarrow 0$
        else $D \leftarrow D + d$
    return newPoints
PATH-LENGTH (A)
    $distance \leftarrow 0$
    for i from 1 to $|A|$ step 1 do
        $distance \leftarrow distance +$ DISTANCE$(A_{i-1}, A_i)$
    return distance
```

Step 2: Generate a vector representation for the gesture. The procedure takes two parameters: *points* are resampled points from Step 1, and *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 (points, oSensitive)
    $centroid \leftarrow$ CENTROID (points)
    $points \leftarrow$ TRANSLATE (points, centroid)
    $indicativeAngle \leftarrow$ ATAN $(points_{0_y}, points_{0_x})$
    if oSensitive then
        $baseOrientation \leftarrow (\pi / 4)$  FLOOR $((indicativeAngle + \pi / 8) / (\pi / 4))$
        $delta \leftarrow baseOrientation - indicativeAngle$
    else  $delta \leftarrow - indicativeAnlge$
    $sum \leftarrow 0$
    foreach point $(x, y)$ in points do
        $newX \leftarrow x$ COS $(delta) - y$ SIN $(delta)$
        $newY \leftarrow y$ COS $(delta) + x$ SIN $(delta)$
        APPEND (vector, newX, newY)
        $sum \leftarrow sum + newX \times newX + newY \times newY$
```

```
magnitude ← SQRT (sum)
foreach e in vector do
    e ← e / magnitude
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)
maxScore ← 0
foreach template in templates do
    distance ← OPTIMAL-COSINE-DISTANCE ($template_{vector}$, vector)
    score ← 1 / distance
if score > maxScore then
    maxScore ← score
    match ← $template_{name}$
return ⟨match, score⟩
```

```
OPTIMAL-COSINE-DISTANCE (vector, vector′)
$a = 0$
$b = 0$
for i from 1 to |vector| Step 2 do
    $a \leftarrow a + vector_i \times vector'_i + vector_{i+1} \times vector'_{i+1}$
    $b \leftarrow b + vector_i \times vector'_{i+1} - vector_{i+1} \times vector'_i$
angle ← ATAN (b/a)
return ACOS (a × COS (angle) + b × SIN (angle))
```

[1] Yang Li, Protractor: a fast and accurate gesture recognizer, *CHI 2010: ACM Conference on Human Factors in Computing Systems*. p. 2169-2172.

[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.