Free-Hand Gesture Recognizer Pseudocode

Elena-Gina Craciun
University Stefan cel Mare of Suceava
Suceava 720229, Romania
ginac@eed.usv.ro

Ionela Rusu
University Stefan cel Mare of Suceava
Suceava 720229, Romania
ionelar@eed.usv.ro

Radu-Daniel Vatavu
University Stefan cel Mare of Suceava
Suceava 720229, Romania
vatavu@eed.usv.ro

Our free-hand recognizer extends the $P gesture recognizer[1] to 3-D gestures performed by the hand in mid-air. In the following pseudocode, HANDPOSE is a structure that exposes x and y coordinates for each finger of the hand, such as delivered by the Leap Motion controller[2]. The pseudocode follows the main structure of the $P++ recognizer.

Recognizer main function. Match C against a set of *templates* by employing the Nearest-Neighbor classification rule.

FREEHAND-RECOGNIZER (List<HANDPOSE> C, TEMPLATES *templates*)

```
n ← 32
NORMALIZE(C, n)
score ← ∞
for each T in templates do
    NORMALIZE(T, n) // should be pre-processed
    d ← min (CLOUD-DISTANCE(C, T), CLOUD-DISTANCE(T, C))
    if score > d then
        score ← d
        result ← T
return ⟨result, score⟩
```

Cloud matching function. Match two HANDPOSE clouds (C and T) by performing one-to-many alignments between their hand poses. Returns the minimum alignment cost.

CLOUD-DISTANCE (List<HANDPOSE> C, List<HANDPOSE> T, int n)

```
matched ← new bool[n]
sum ← 0
// match hand poses from cloud C with poses from T; 1-to-many matchings allowed
for i = 1 to n do
    min ← ∞
    for j = 1 to n do
        d ← HANDPOSE-DISTANCE(C_i, T_j)
        if d < min then
            min ← d
            index ← j
    matched[index] ← true
    sum ← sum + min
// match remaining hand poses T with poses from C; 1-to-many matchings allowed
for each j such that not matched[j] do
    min ← ∞
    for i = 1 to n do
        d ← HANDPOSE-DISTANCE(C_i, T_j)
        if d < min then min ← d
    sum ← sum + min
return sum
```

The following pseudocode addresses gesture preprocessing (or normalization) which includes resampling, scaling with shape preservation, and translation to origin.

Gesture normalization. Gesture points are resampled, scaled with shape preservation, and translated to origin.

Normalize (List<HANDPOSE> *gesture*, int n)

```
gesture ← RESAMPLE(gesture, n)
SCALE(gesture)
TRANSLATE-TO-ORIGIN(gesture, n)
```

Gesture resampling. Resample a *gesture* path into n evenly spaced hand poses. We use $n = 32$.

RESAMPLE (List<HANDPOSE> *gesture*, int n)

```
I ← PATH-LENGTH(gesture) / (n − 1)
D ← 0
newGesture ← gesture_0
for each hand_i in gesture such that i ≥ 1 do
    d ← HANDPOSE-DISTANCE(hand_{i−1}, hand_i)
    if (D + d) ≥ I then
        // interpolate two hand poses
        q ← hand_{i−1} + ((I − D)/d) · (hand_i − hand_{i−1})
        APPEND(newGesture, q)
        INSERT(newGesture, i, q) // q will be the next hand_i
        D ← 0
    else D ← D + d
return newGesture
```

PATH-LENGTH (List<HANDPOSE> *gesture*)

```
d ← 0
for each hand_i in gesture such that i ≥ 1 do
    d ← d + HANDPOSE-DISTANCE(hand_{i−1}, hand_i)
return d
```

Gesture rescale. Rescale *gesture* with shape preservation so that the resulting bounding box will be $\subseteq [0..1] \times [0..1] \times [0..1]$.

SCALE (List<HANDPOSE> *gesture*)

```
x_min ← ∞, x_max ← 0, y_min ← ∞, y_max ← 0
for each hand in gesture do
    for i = 1 to hand.numFingers do
        x_min ← MIN(x_min, hand.x_i)
        y_min ← MIN(y_min, hand.y_i)
        x_max ← MAX(x_max, hand.x_i)
        y_max ← MAX(y_max, hand.y_i)
scale ← MAX(x_max − x_min, y_max − y_min)
for each hand in gesture do
    for i = 1 to hand.numFingers do
        hand.x_i ← (hand.x_i − x_min)/scale
        hand.y_i ← (hand.y_i − y_min)/scale
```

Translate. Translate *gesture* to the origin.

TRANSLATE-TO-ORIGIN (List<HANDPOSE> *gesture*, int n)

```
c ← (0, 0) // will contain centroid
for each hand in gesture do
    for i = 1 to hand.numFingers do
        c ← (c.x + hand.x_i, c.y + hand.y_i)
c ← (c.x/n/numFingers, c.y/n/numFingers)
for each hand in gesture do
    for i = 1 to hand.numFingers do
        hand.x_i ← hand.x_i − c.x
        hand.y_i ← hand.y_i − c.y
```

Hand pose distance. Computes the distance between two hand poses as the sum of Euclidean distances between their fingers' coordinates.

HANDPOSE-DISTANCE (HANDPOSE a, HANDPOSE b)

```
d ← 0
for i = 1 to a.numFingers do
    d ← d + ||a_i − b_i||
return d
```

[1] http://dx.doi.org/10.1145/2388676.2388732
[2] https://www.leapmotion.com/

Project no. PN-II-RU-TE-2014-4-1187, financed by UEFISCDI, Romania.