Fingerprint template matching

Marcel Portillo

Biel, HS 2015

```
RecordHeader of Isotemplate   1
    Image Width:          288
    Image Height:         384
    X Resolution:         197
    Y Resolution:         197
    N Views:              1
  FingerViewHeader IndexNr   0
    fingPos:              0
    nView:                0
    imprType:             0
    fingQuality:          0
    nMinutiae:            36
  MinutiaIndex:       0
    minType:              2
    xCoord:               173
    yCoord:               79
    minAngle:             244
    minQuality:           0
  MinutiaIndex:       1
    minType:              2
    xCoord:               170
    yCoord:               89
    minAngle:             233
    minQuality:           0
```

Überblick

1 Problemstellung

In einem Textfile sind nach ISO Standard Fingerprint Templates abgelegt. Es gilt nun herauszufinden welche denselben Finger darstellen. Dazu werden die Minutiae verglichen. Nun können aber bei den Daten Verschiebungen, unterschiedliche Bildausschnitte und kleinere Rotationen vorkommen. Deshalb ist nur aufgrund der Koordinaten nicht abzusehen, ob ein Minutia einem anderen entspricht.
Die Rohdaten müssen geparst und aufbereitet werden um überhaupt vergleichbar zu sein. Durch Anwendung verschiedener Operationen werden die Daten zweier Fingerprints zur Überprüfung vorverarbeitet.

2 Vergleichsalgorithmus

2.1 Einleitung

Der Lösungsansatz sieht einen Vergleich durch Vektoren vor. Die Minutiae zu vergleichen ist ohne weiteres nicht möglich. Zuerst müssten Translationen und Rotationen eliminiert werden. Diese alleine aufgrund der Minutiae bestimmen zu können stellt ein schwieriges Unterfangen dar. Wären zwei Minutia gegeben, von denen man wüsste, dass sie auf beiden Bildern identisch sind, wäre es ein leichtes die Verschiebungen zu bestimmen. Da es sich aber um zwei verschiedene Finger handeln kann ist dies gar nicht möglich. Anstatt mich mit den Minutiae zu beschäftigen habe ich den gesamten Algorithmus auf Vektoren basiert.

2.2 Programmiersprache

Python als Interpreter Sprache schien mir eine geeignete Wahl zu sein. Da es keine Variablen Deklarationen und Klammern gibt, bleibt der Code kurz und übersichtlich. Ausserdem stehen einige mathematische Bibliotheken zur Verfügung. Read und Write Operationen auf Dateiebene produzieren keinen Overhead.

2.3 Parsen

Das vorgegebene ISO Template wird von einer Pythonklasse geparst und in einer OOP mässigen Datenstruktur abgelegt. So können die einzelnen Fingerprints mit den dazugehörigen Minutae bequem vom Code aus verwaltet werden und es ist kein weiterer Dateizugriff erforderlich. Alle werden dabei ins Memory geladen.

2.4 Vergleichen

2.4.1 Matrizen bilden

Der Vergleichsalgorithmus läuft in verschiedenen Schritten mit verschiedenen Iterationsstufen ab. Zuerst müssen die Minutiae als Vektoren dargestellt werden. Dazu wird eine Matrize aufbereitet, die alle von den Minutia aufgespannten Vektoren beinhaltet. Diese Matrize hat die Grösse n * (n -1) auf 4. Alle Vektoren die bloss die Gegenvektoren eines anderen darstellen werden aussortiert. Nach diesem Schritt hat die Matrize noch die Grösse n * (n – 1) / 2 auf 4.

2.4.2 Rotation

Für jeden Winkelgrad im Definitionsbereich (bestimmt durch Parameter) wird der Vergleichsalgorithmus aufgerufen und werden die Vektoren verglichen. Diejenige Rotation, die am meisten echte Übereinstimmungen gefunden hat wird gewählt. Dabei wird bei jedem Winkel über die Vektoren der Matrix des ersten Fingerprints iteriert. Jeder dieser Vektoren wird um den Winkel rotiert und anschliessend mit den Vektoren der Matrix des zweiten Fingerprints verglichen.

2.4.3 Translation

Beim Vergleich des rotierten Vektors mit den Vektoren aus der zweiten Matrix müssen auch Verschiebungen berücksichtigt werden. Ist der erste Vektor und einer der Vergleichs Vektoren linear abhängig, so sind die beiden kollinear. Es könnte sich also dabei um eine Verschiebung handeln.

2.4.4 Eine reale Übereinstimmung

Es kann sein, dass zwei Minutiae aus der einen Matrix einen kollinearen Vektor zu dem Vektor zweier anderer Minutiae von der anderen Matrix aufspannen, jedoch handelt es sich nicht um den gleichen Fingerprint. Damit verhindert wird, dass die Punkte jeweils an vollkommen verschiedenen Orten im Bild liegen wird ein maximaler Abstand definiert. Jede der Übereinstimmungen von Vektoren der Matrizen muss überprüft werden. Liegen die Anfangs- und Endpunkte des einen Vektors zu weit von den Anfangs- und Endpunkten des anderen Vektors, so handelt es sich nicht um eine richtige Übereinstimmung. Ausserdem wird der Minutiatyp überprüft. Dieser muss identisch sein.

2.4.5 Trotzdem Falsch

Auch die Überprüfung der realen Übereinstimmung kann nicht verhindern, dass ein Vektor mit einem falschen Vektor übereinstimmt. Dies kann passieren falls die Minutae zu nahe beieinanderliegen und eine Übereinstimmung anzeigen, obwohl es sich dabei um verschiedene Minutiae von verschiedenen Fingerprints handelt. Deshalb ist der Schwellenwert wichtig. Es wird eine Mindestzahl von realen Übereinstimmungen festgelegt, die auf jeden Fall erfüllt sein müssen. Bei übereinstimmenden Fingerprints wird die Anzahl gefundener Übereinstimmungen sicherlich höher sein.

2.5 Laufzeitbetrachtung

Es gibt 60 Fingerprint Templates in der Textdatei. Im Schnitt hat jedes davon 30 Minutiae. Um zwei Fingerprints zu vergleichen wird die Matrix aufgebaut: 30 * 29 / 2 = 435 Vektoren. Diese beiden Matrizen müssen nun gegeneinander geprüft werden. Jeder Vektor wird mit jedem Vektor abgeglichen. Dies führt zu 4352 = 189225 Operationen. Für jeden Iterationsschritt bei der Rotationsberechnung müssen alle diese Operationen ein weiteres Mal durchgeführt werden. Dies bedeutet also 189225rotCount Berechnungen. Der Algorithmus weisst also eine Laufzeit von O(nm4) auf, exakter O(n4m/4m). Wobei m die Anzahl Rotationschritte darstellt. Um alle Fingerprints zu vergleichen (auch dieselben mit sich selbst) müssen 60 * 60 / 2 = 1800 Operationen angewandt werden. Dies bedeutet einen totalen Aufwand von (189225rotCount)1800 Berechnungen. Selbst für eine moderne CPU ist dies doch ein bisschen viel.
Dieser Algorithmus bevorzugt Einfachheit, hat aber dafür eine längere Laufzeit. Für eine Überprüfung zweier Fingerprints reicht es aus. Allerdings für mehrere Überprüfungen ist es nicht geeignet (Fingerprint gegen eine Datenbank laufen lassen).

3 Resultate

3.1 Vergleichswerte

Mit entsprechenden Parametern wurde die Testbestimmung erfüllt. Die in der Aufgabestellung enthaltenen Entscheide konnten reproduziert werden. Dazu wurden Rotationen bis zu 4 Grad in 1er Schritten überprüft. Die Minutiae der realen Übereinstimmung durften bis zu 50 px abweichen. Dafür aber mussten mindestens 10 übereinstimmende Vektoren gefunden werden. Das von diesen unzähligen Überprüfungen immer nur so wenige reale Übereinstimmungen gefunden werden liegt an der Präzision der Überprüfung der Translation. Auf bis zu zwei Dezimalstellen genau muss ein Vektor ein Vielfaches eines anderen sein.

3.2 Güte

Es ist schwer zu sagen, wie die Resultate zustande kommen. Um Aussagen machen zu können müssten die gefundenen Vektoren und Minutiae z.B . visuell ausgewertet werden. Es wäre auch durchaus denkbar, dass der gesamte Algorithmus nur auf Zufall und Glück basiert. Vermutlich liegen viele Minutiae in einem 50px Radius (Bildgrösse 288 x 384). Die 50px Radius umspannen ein 11 des gesamten Bildes. Die Chance dass also ein Vektor innerhalb dieser Toleranz mit einem anderen Vektor übereinstimmt ist relativ gross, auch wenn es sich um Finger von verschiedenen Personen handelt. Je nachdem ob zwei Minutiae Paare relativ exakt gleich liegen wird ein Treffer gefunden, dies könnte auch tatsächlich völlig willkürlich sein.

3.3 Optimierung

Anhand einiger zufälliger Vektoren eine mögliche Rotation und Translation ermitteln und anschliessend auf alle Vektoren der Matrize anwenden. Anschliessen könnten tatsächlich die Minutiae eins zu eins verglichen werden. Ausserdem könnte ein Verfahren erstellt werden, dass die zu vergleichenden Vektoren nach einem intelligenten Schema auswählt (Minutia Typ, Minutia Winkel, Quadrant). Allerdings müsste um die Rotation zu eliminieren ein Rotationszentrum gefunden werden können.
Eine Programmiersprache die näher an der Hardware arbeitet ist für diesen Zweck besser geeignet. Python als Interpreter Sprache ist für zeitintensive Rechneraufgaben vermutlich weniger performant als beispielsweise C.
Die Vektorvergleiche und Matrizen wurden alle ausprogrammiert. Mathebibliotheken die solche Funktionen bereits anbieten sind leistungsoptimiert und teilweise schneller. Ebenfalls könnte der bestehende Code optimiert werden. Es gibt bestimmt mathematische Verfahren, die einen Vergleich über die gesamte Matrize laufen lassen und automatisch alle linear abhängigen Vektoren zurückgibt in einem Schritt. Multithreading wäre ebenfalls eine Option. Die zu vergleichenden Vektoren einer Matrize könnten in vier Gruppen aufgeteilt werden und jede davon auf einem anderen Kern verglichen werden.