Bachelor Project 2012

Analysis of the Euclidean Feature Transform algorithm

Sebastian Verschoor

Supervisors:

› prof. dr. Gerard R. Renardel de Lavalette

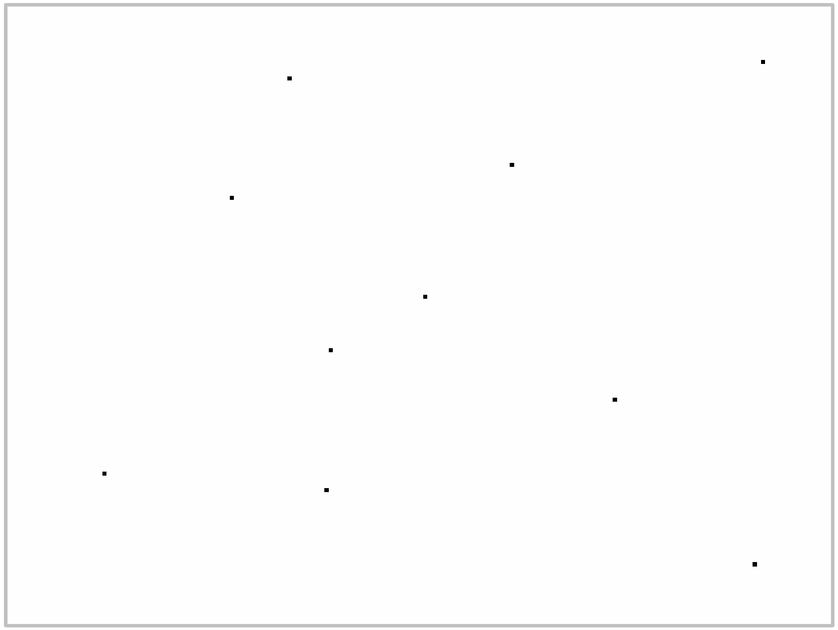
› prof. dr. Wim H. Hesselink

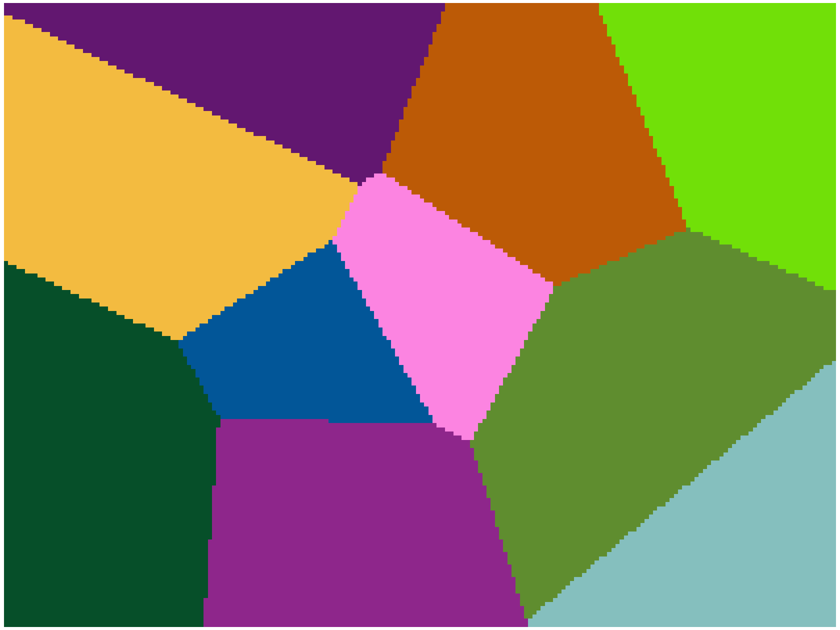
Table of Contents

The project goal

- The goal of this bachelor project is to mechanically verify (or even disprove) that the algorithm as posed by Hesselink [1] correctly calculates the Euclidean Feature Transform (EFT), and does so in linear time complexity.
- Mechanical Verification > Mathematical Proof

The Euclidean Feature Transform (EFT)





The EFT algorithm

- The algorithm uses some clever tricks
  - Iterating the dimensions, using the same algorithm for solving the base case and the inductive step

- Reduces the problem to finding the one-dimensional EFT
- $O(n)$ (n number of "pixels")

The EFT algorithm

```
OneFT(n, h):
    q ← 0; t[0] ← 0; at[0] ← 0
    for (k ← 1; k < n; k++)
        while (q ≥ 0 ∧ f (t[q], at[q]) > f (t[q], k))
            q ← q - 1
        if (q < 0)
            q ← 0; at[0] ← k
        else
            w ← 1 + g(at[q], k)
            if (w < n)
                q ← q+1
                t[q] ← w; at[q] ← k
    t[q+1] ← n; at[q+1] ← n − 1
    for (j ← 0; j = q; j++)
        x1 ← t[j+1] − 1
        for (x ← t[j]; x = x1; x++)
            FT[x] ← {at[j]}
        for (p ← at[j] + 1; p = at[j+1]; p++)
            if (f (x1, p) = f (x1, at[j]))
                FT[x1] ← FT[x1] ∪ {p}
```

Mechanical Verification

SRI International

PVS

Welcome to the PVS Specification and Verification System

- Prototype Verification System (PVS 5.0)
  - SRI International, Computer Science Laboratory
- Specification Language
- Interactive Prover

PVS Specification Language

- Based upon simple typed logic
- Formal specification of the problem
  - Types
  - Definitions
  - Theorems / Lemmas

PVS Prover

- Proof obligation
  - Logical sentence:
    $P_0 \wedge P_1 \wedge \ldots \wedge P_m \Rightarrow Q_0 \vee Q_1 \vee \ldots \vee Q_n$
- Proof commands
  - Rewrite proof obligation to a logical equivalent statement

- The Prover does not prove anything!
  - It is merely keeps a "smart" administration

PVS Prover - Example

Program Correctness

- programs.pvs
  - Hoare-Triplets:
    - {P} S {Q}
  - While loops
    - 5 steps
    - Prove correctness and termination

Project Progress (done)

- Learning PVS
  - Basics of the master course Automated Reasoning
- Understanding the algorithm
- Verified the mathematics
- The algorithm
  - Proved on paper
  - Specified in PVS
- 118 theorems/lemmas
  - 91 proven

Project Progress (todo)

- Prove the algorithm
  - With PVS
- Optional: prove the mathematics behind iterating the dimensions
- Write thesis

Evaluation

- Mechanically verifying a problem does not *result* in a deeper understanding of a problem
    - It does *require* a full understanding of the problem
- PVS is a great tool for proving complex mathematical theorems
    - But, often it feels like you do a lot of trivial work that could somehow be automated

Thank you for your attention

Are there any questions?

References

[1] W. H. Hesselink, "Distance transforms and feature transform sets," May 2009. An extension and modification of the IPL paper.