A Distributed Particle Filter Implementation for Tracking in a Wireless Sensor Network

Jesse Read, Katrin Achutegui, Joaquín Míguez

Universidad Carlos III de Madrid.

January 22, 2013

Mote Hardware (the iMote2)

Processor:

- 13–104 MHz CPU
  but no native support for floating point numbers!
- 32 MB SDRAM

Network:

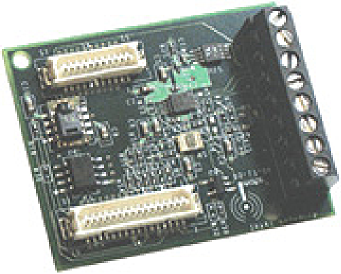
- 802.15.4 radio transceiver
- 250 Kb/s

Power:

- 3xAAA batteries (days – months)

Basic Sensor board:

- Light sensor (among others)



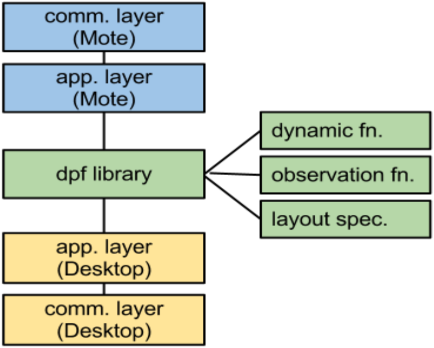
A Framework for Developing `iMote2` applications

Software:

- `TinyOS` operating system; programming in `NesC`;
  this is a difficult setup on the `iMote2` (no simulator, etc)!

Our framework:

- a *dpf library*: written in `C`, and
- an *applications layer* which connects dpf functions with
- a *communications layer*: interface with radio (stemmed from work with the UPM)



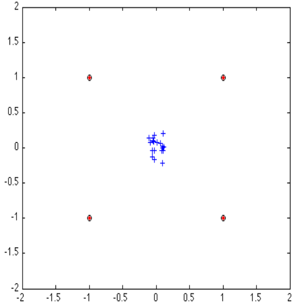
We develop and simulate the dpf on a Desktop PC (*app*/*comm*. layers written in `C`), then install it directly to the motes to work with the 'real' layers (written in `NesC`) deployed in a WSN.

Particle Filters for Tracking in WSNs

Given noisy *observations* (sensor measurements), particle filters can recursively estimate the *state* (position and velocity) of a target.

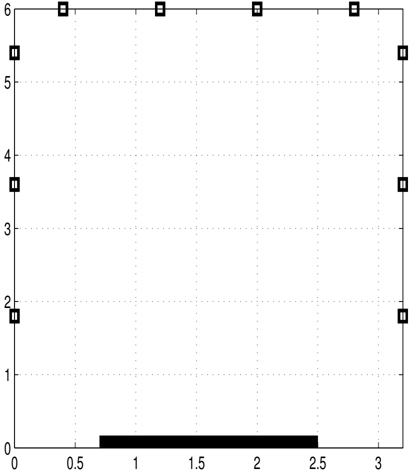
Particles . . .

1. move like a target (*dynamic fn.*), explore the *space* (*layout spec.*)
2. weighted by how likely they are given the observations (*observation fn.*)
3. resampled (replicated) according to weight



The target is estimated as weighted-average of particles at any time step.

A Real-world Setup



- 3.2×6.0 metres, indoors
- single light source (a window)
- layout of 10 `iMote2` motes
- light sensor observations $\mathbf{y}_t$
  e.g., $[0, 1, 1, 0, 0, 0, 0, 0, 1, 0]$
- to estimate position $\in \mathbb{R}^2$

Implementing a Particle Filter

Problem:

- PF with 100 particles on Intel Xeon 3.16GHz CPU: 75000 steps/min
- but PF with 100 particles on `iMote2` mote: 17/min (too slow!) (although network capable of $\approx$ 300 steps/min)

Implementing a Particle Filter

Problem:

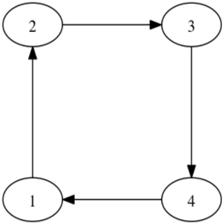
- PF with 100 particles on Intel Xeon 3.16GHz CPU: 75000 steps/min
- but PF with 100 particles on `iMote2` mote: 17/min (too slow!) (although network capable of $\approx$ 300 steps/min)

Solution: a Distributed Particle Filter

- N processing elements (PEs), running PFs in parallel, $100/N$ particles each (100 in total).

At each timestep, each PE:

1. receives a particle from another PE
2. for all its particles:
   1. move
   2. weight
   3. resample
3. send best particle to another PE

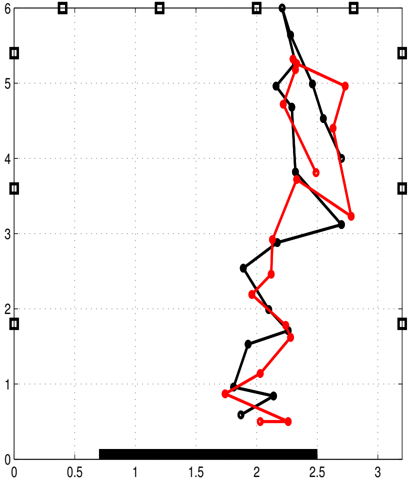


where $N = 4$

+ At each step, *all* motes/sensors broadcast their observations.

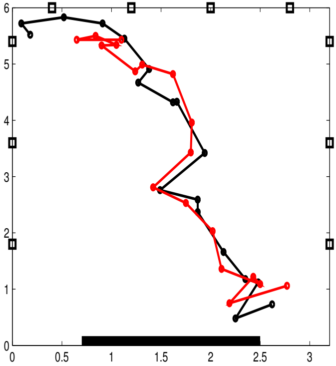
Simulations (fitted with real data)

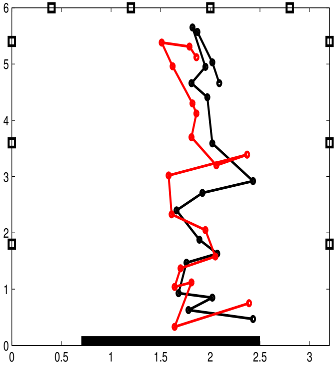
Estimating (**red**) a simulated target trajectory (**black**). **[**`animation`**]**



Simulations (fitted with real data)

Estimating (**red**) a simulated target trajectory (**black**). Two more.

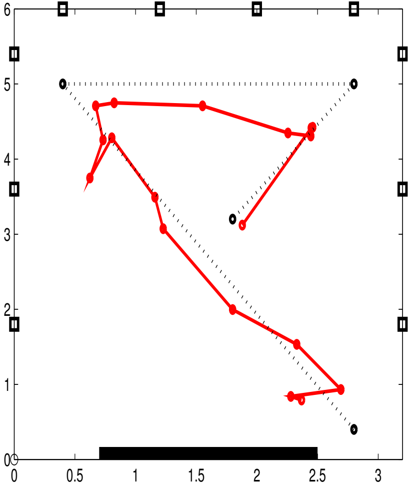




- DPF ($N = 4$) error of 0.5115 across 100 simulations (0.4901 for CPF)
- running at 1 timestep / sec. (what we have achieved in experiments)

Experiments - Real Trajectory

Katrin walks a path (dashed line), and we track her in real time.



Testbed Performance Analysis

Table: Performance per step for 10 nodes, N PEs with $100/N$ particles each.

	CPF	$N = 2$	$N = 4^\star$	$N = 8$
Seconds/timestep (excl. net.)	3.37	1.51	0.73	0.26
Packets/timestep†	9	10	10	10
Bytes/timestep‡	9	98	178	338

$\star$ our main deployment (used for simulations + experiments)

$\dagger$ we combine observations + donated particles into a single packet

$\ddagger$ excluding overhead. The motes' radio is rated at 31,250 B/s.

Summary so far

- A realworld deployment of a DPF for real-time tracking in a WSN
- using light-sensor measurements for tracking
- a framework for implementing particle filters in WSNs
- a modular DPF library
- results in our indoor scenario are good

We have submitted a journal article with our results.

Limitations

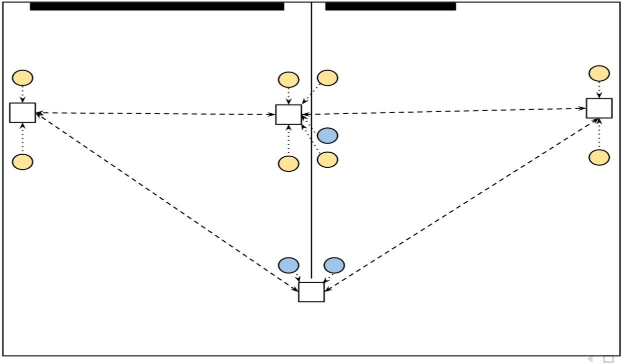
- Scaling up; $N = 100 \Leftrightarrow 40KB/timestep (\gg 100KB/s)$
- Network disruption (failing node, etc) not dealt with
- Constant use drains battery quickly (< 2 days)
- Development+Deployment still very slow/painstaking on the motes
- Until now ... single target only, only light-sensor observations

Work in Progress

- Dedicated processing elements (PEs); "beagleboard" (left)
  - 7.6 cm $\times$ 7.6 cm
  - 600 MHz, 128 MB RAM; standard Linux installation
  - processes observations, makes estimations
- Simpler sensing elements (SEs); "telosb" (right)
  - 8.2 cm $\times$ 3.3 cm.
  - low spec, 10 KB RAM; `TinyOS` again
  - relays observations to nearest PE

Work in Progress

- Fusion of observations from different kinds of sensors (light, acoustic intensity, RSSI)
- Most communications between PEs
- Deployment in other/larger scenarios, with a larger network
- Multi-target tracking



Thank you! Any Questions?

P.S. if anybody has an algorithm they would like to try in our WSN testbed, let us know!