A WSN Testbed for Distributed Signal Processing

Jesse Read, Katrin Achutegui, Joaquín Míguez

Universidad Carlos III de Madrid.

July 5th, 2011

Index

Outline

1. Hardware for WSNs
2. A Communications Layer
3. Algorithms for WSNs
4. Summary and Future Work

Mote Hardware (iMote2)

Processor:

- 13-416MHz Intel processor
- 256 KB SRAM
- 32 MB SDRAM
- 32 MB Flash Memory

Network:

- 802.15.4 radio transceiver
- 250 Kb/s

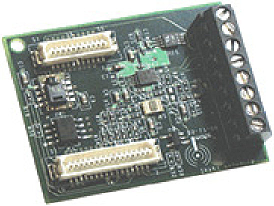
Power:

- 3xAAA batteries (days – months)

Sensor Hardware

Basic Sensor Board (20 units):

- 3-axis accelerometer
- Light/IR
- Temperature, Humidity

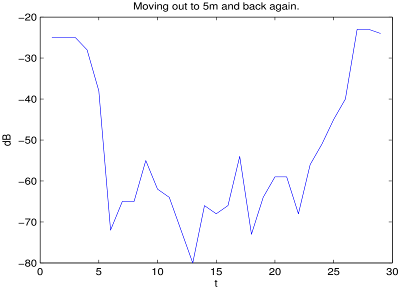


Media Sensor Board (10 units):

- Color Video Camera
- PIR Motion detector
- Microphone + speaker

Experiments

- RSSI data measurements
  - range down to around -100 dB
  - very noisy in ordinary environments
- Light sensor measurements
  - range up to around 3000 '*units*'
  - very robust; sensitive when an object comes *directly* between the sensor and the light source (especially under natural light)
- Audio and still images

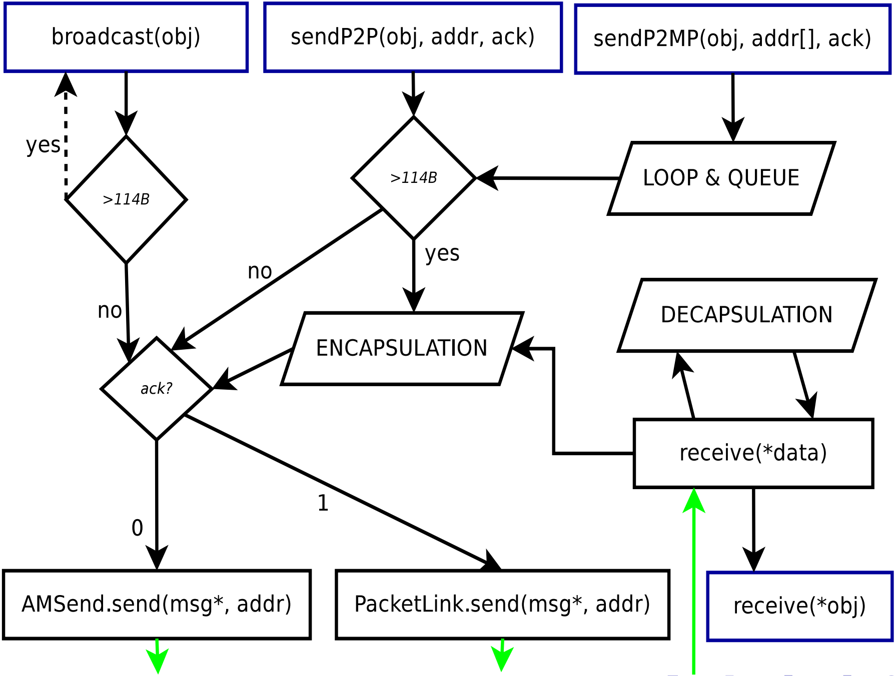


Outline

A Communications Layer

A communications layer (*joint work with UPM, US*):

- for rapid development of WSN applications
- using as much standard/existing functionality as possible
- able to send any objects (structs, data arrays, floating point values, etc)
  - point-to-point
  - point-to-multipoint
  - broadcast
- able to set different *quality of service* levels (i.e., ACKs)
- encapsulation of large objects (the iMote2s are restricted to a *maximum of 114 Bytes / packet*!)
  - two motes are locked for transmitting encapsulated messages
  - standard messages ($< 114B$) given priority


broadcast(obj)
sendP2P(obj, addr, ack)
sendP2MP(obj, addr[], ack)
yes
>114B
>114B
LOOP & QUEUE
yes
no
no
DECAPSULATION
ENCAPSULATION
ack?
receive(*data)
1
0
AMSend.send(msg*, addr)
PacketLink.send(msg*, addr)
receive(*obj)

Example

```
Comm<StateEst_t> as StateEstComm;
Comm<uint8_t> as CommandComm;

event msg_t *StateEstComm.receive(am_addr_t addr,
        object_t *payload) {
    stateEstimate = (StateEst_t *)payload;
    x = stateEstimate->x;
    y = stateEstimate->y;
    ...
    StateEstComm.sendP2P(myStateEstimate,addr);
}

task void sendTask() {
    ...
    CommandComm.broadcast(CMD_DONE);
}
```

Outline

Particle filters

- Particle filters are **numerical simulation methods** based on Bayesian theory and the **importance sampling** methodology.
- Given noisy *observations* (e.g. sensor measurements), they can recursively estimate the *state* (e.g., position and velocity) of a target.
- We have demonstrated that these methods perform well with real data from wireless sensor networks (WSNs) in a tracking problem (*work performed in collaboration with UDC*).
- Why particle filters?
    - interesting / suitable problem for WSNs
    - sufficiently technical / demanding: if we can implement particle filters, we can implement much more!

Centralized particle filter for tracking.

Centralized particle filter for tracking.

1. **Initialization**, at $t = 0$:
   - Draw random samples from the prior distribution; assign them equal weight.
2. **Recursive step**, for $t > 0$:
   - **Sampling step**: draw samples randomly from the *importance function*.
   - **Importance step**: update the weights with the *observations* (sensor measurements)
   - **Estimation**: estimate the *state* (position, velocity)
   - **Selection/resampling step**: Replicate "good" particles and delete "bad" ones.

Distributed Particle filters.

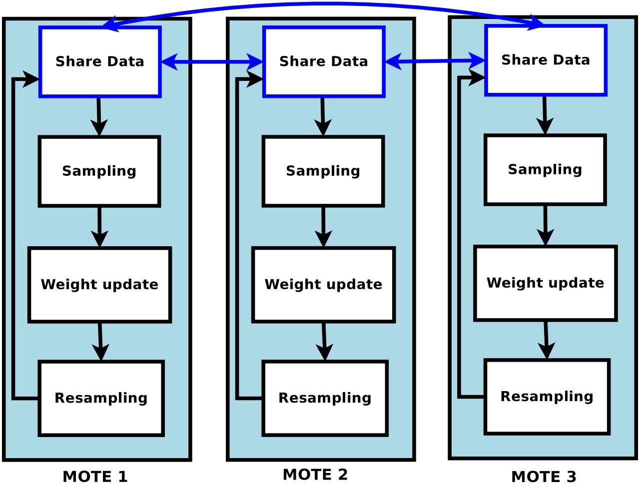
- *Each mote* broadcasts 10-20% of its *particles* (and their *weights*) + its *observation*.
- *All motes* run a particle filter (in parallel) and compute their **local estimate**.
- The **global estimate** = weighted mean of the local estimates (each mote must broadcast its local estimate + weight).

Distributed particle filter for tracking.

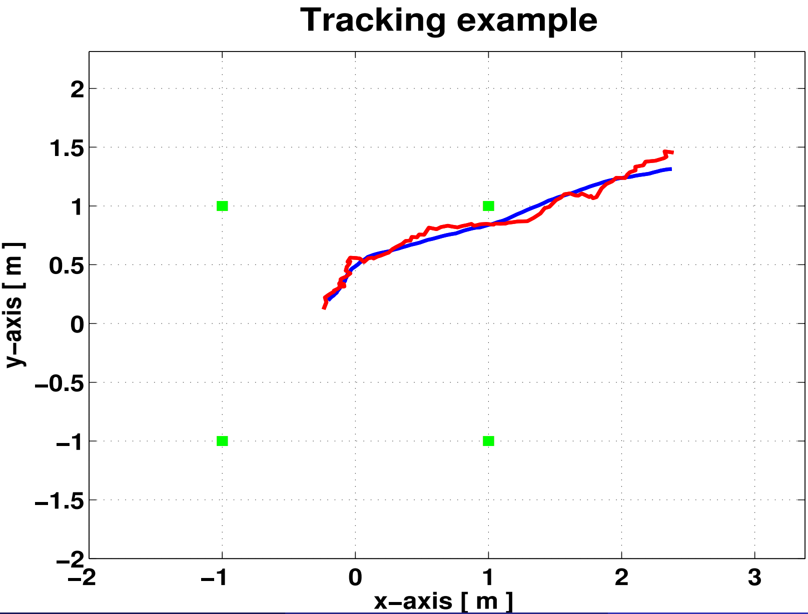
Distributed particle filter for tracking.

1. **Initialization**, at $t = 0$:
   - Draw random samples from the prior; assign them an equal weight.
2. **Recursive step**, for $t > 0$:
   - For each mote:
     - **Information passing step**: broadcast *particles* (with *weights*) + *observations* [+ optionally local estimate].
       (10-20% of the *best* (highest weight) particles are shared)
   - For all motes (in parallel):
     - **Sampling step**: draw samples randomly from the importance function.
     - **Importance step**: update the weights with the *observations* (sensor measurements).
     - **Estimation**: *local* estimation of the *state* (position, velocity).
     - **Selection/resampling step**: Replicate "good" particles; and delete "bad" ones
   - For any mote (when we need an estimate):
     - gather local estimates and **compute global estimate**

Distributed Particle Filter Example



Example Tracking Problem



Implementation

Particle filter implementation (NesC/C, TinyOS)

- 4 sensors (simulated; centralised)
- 100 particles

Performance:

- Network: $\approx$ 5 packets/mote/sec (no ACKs)
  $=$ 0.20 seconds/timestep
- Processing: $\approx$ 9 seconds/timestep

Remarks:

- processing is the main bottleneck ($<$ 7 timesteps/*min*)!
- some optimisations possible (e.g., fewer floating point values); but
- better to distribute processing.

Future Implementation

Distributed particle filter (planned):

- 4 motes
- 25 particles *per mote* (=100 particles)
- 5 particles (20%) shared between

Performance:

- Network: one packet ($114B$) may hold a *sensor observation + 10 particles with weights + local estimate* ($<$ 1 sec/timestep)
- Processing: $\approx$ 9 seconds/timestep / 4 motes = 2.25 seconds/timestep

Remarks:

- now 24 timesteps/*min*!
- network time *increases* with more motes
- processing time *decreases* with more motes
  - not linearly if we wish to maintain accuracy, but significantly!

Outline

Summary and Future Work

- iMote2 Hardware
  - RSSI and light sensor measurements
  - future work to use audio and image data
- Communications layer
  - for facilitating implementations
- Particle filters for tracking
  - processing is a major bottleneck
  - future implementation of distributed particle filters
  - (and a library of functions for signal processing algorithms)

References.

- Achutegui, K.; Martino, L.; Rodas, J.; Escudero, C.J.; Miguez, J., "A multi-model particle filtering algorithm for indoor tracking of mobile terminals using RSS data," *IEEE Control Applications, (CCA) and Intelligent Control, (ISIC)*, 2009
- Achutegui, K.; Rodas, J.; Escudero, C.J.; Míguez, J., "A model-switching sequential Monte Carlo algorithm for indoor tracking with experimental RSS data," *Indoor Positioning and Indoor Navigation (IPIN)*, 2010
- F. Gustaffson and F. Gunnarsson, "Mobile positioning using wireless networks" *IEEE Signal Processing Magazine*, vol. 22, no. 4, pp. 41-53, July 2005.
- A. Doucet, N. de Freitas and N. Gordon, Eds. "Sequential Monte Carlo Methods in Practice", New York (USA): Springer, 2001.
- T.S. Rappaport, "Wireless Communications: Principles and Practice (2nd edition)", Prentice-Hall, 2001.