

From Tracking Pixels to Tracking Predicates

Leonidas J. Guibas

Xerox PARC and Stanford University



Xerox Palo Alto Research Center

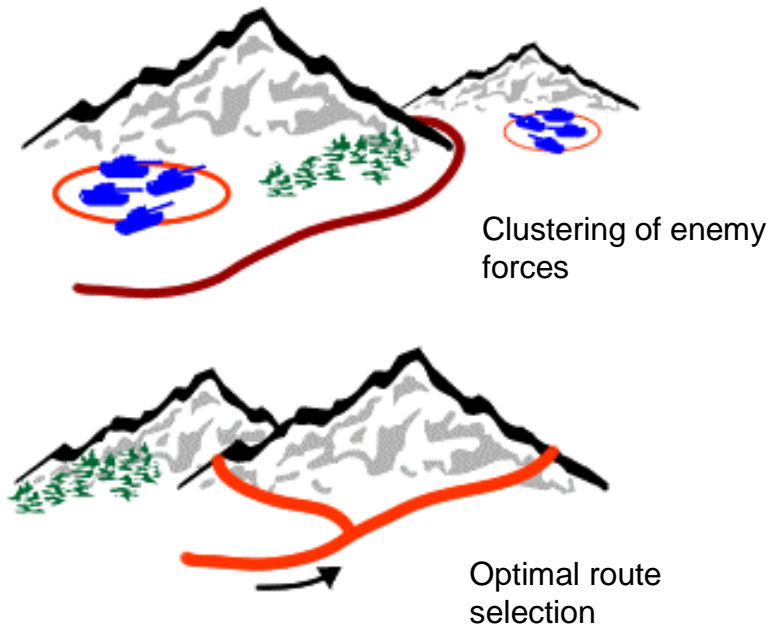
parc



Sensing for Reasoning and Acting

A system of collaborating sensors should provide data that can:

- Lead to high-level understanding of a situation
- Support efficient decision-making



Dealing with Motion and Change

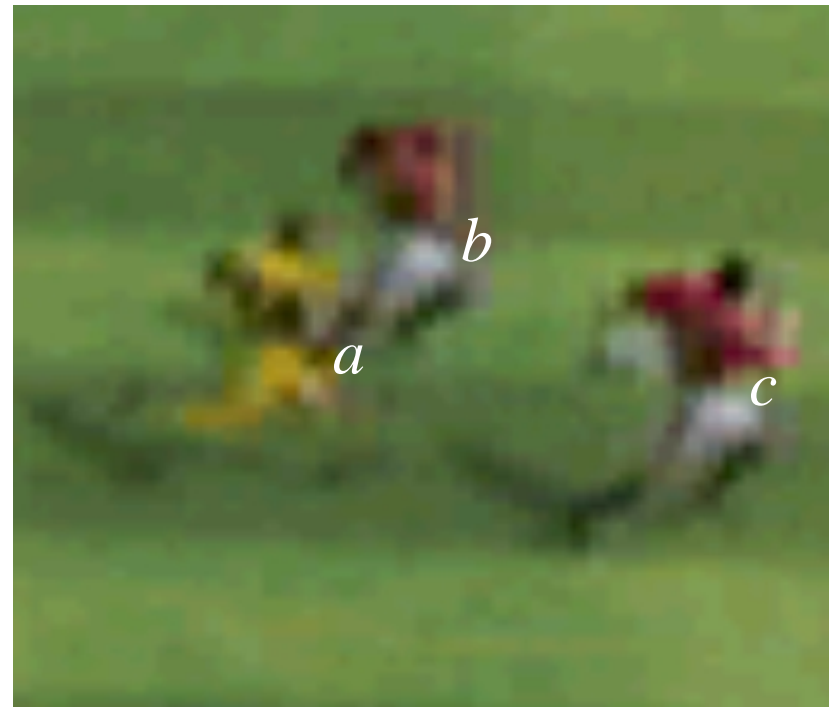
How can we continuously track such **high-level attributes** of interest, without continuous re-computation from scratch?

Key ideas:

- Use sensors to **track** certain **elementary relations** among the objects
- From these **collaboratively compute** the values of the desired attributes
- Update the attribute value **incrementally** as objects move

Tracking Relations vs. Objects

- Tracking relations can be more robust than tracking exact object positions or poses
- When relations become unsupportable, alternate relations may do just as well



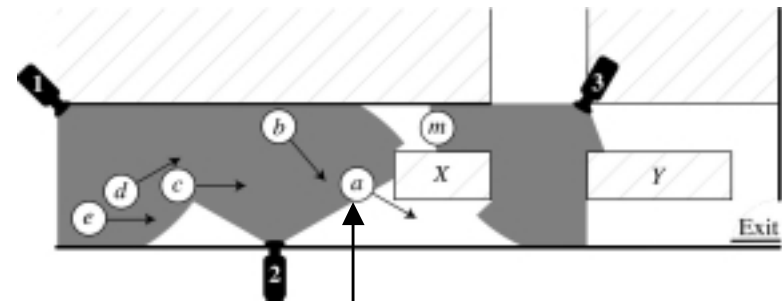
An Example: Leader in the Corridor

Cameras observe individuals running towards the exit:

Camera 1 observes “*a* ahead-of *b*”, “*b* ahead-of *c*”

Camera 2 observes “*c* ahead-of *d*”, “*d* ahead-of *e*”

System conclusion: *a* is the leader



| |
|----------------------------|
| <i>a</i> ahead-of <i>b</i> |
| <i>b</i> ahead-of <i>c</i> |
| <i>c</i> ahead-of <i>d</i> |
| <i>d</i> ahead-of <i>e</i> |

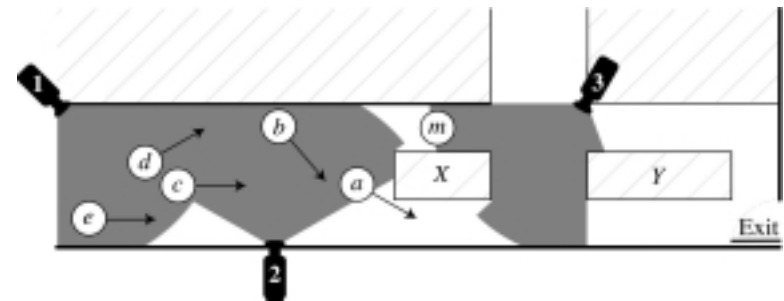
Tracking the Leader under Motion

Suppose d moves ahead to overtake c

Camera 1 can no longer support the relation “ c ahead-of d ”

Camera 1, however, can support the relation “ b ahead-of d ”

It follows that a is still the leader



| |
|------------------|
| a ahead-of b |
| b ahead-of c |
| b ahead-of d |
| d ahead-of e |

Choosing which Relations to Track

We need to choose sets of elementary relations to track that:

- vary “smoothly” as the objects move
- always make the computation of the attribute of interest fast

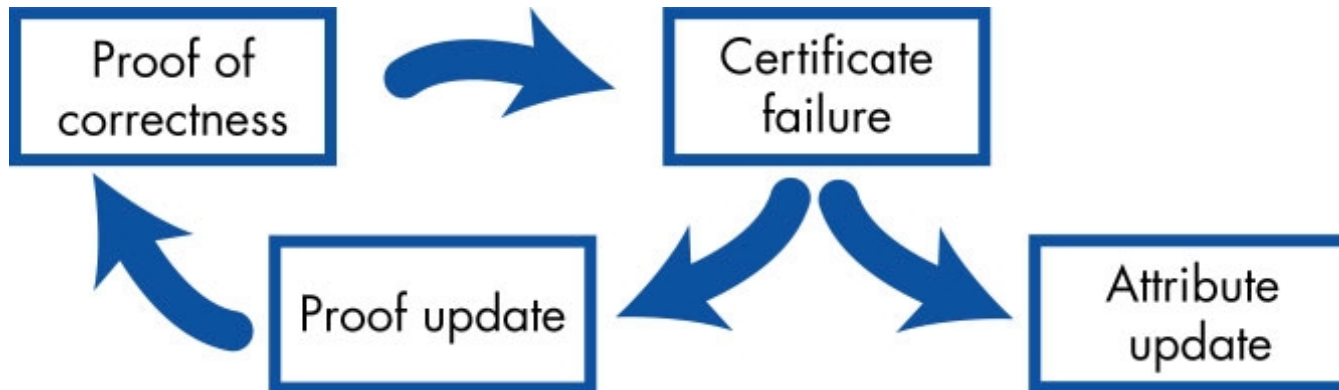
Thus at all times we maintain an **assertion cache** about the state of the world. This cache is continuously updated as assertions fail, or become unsupportable by the sensors.

Kinetic Data Structures (KDS)

- A KDS for an attribute of interest is an **easily repairable set** of elementary relations (the **certificates**) that allows an **easy computation** of the attribute of interest
- At each certificate failure, the KDS procedure repairs the assertion set
- At all times, the certificates mathematically prove the validity of the attribute computation

A KDS is a proof animated through time.

The Eternal KDS Loop



Example: Kinetic Mobile Clustering

Clustering is fundamental for sensor data collection and summarization, non-local communication and routing, etc.

- Cluster mobile nodes (sensors) into groups of a certain geometric size (e.g., determined by communication range)
- Minimize the number of clusters used
- Minimize the number of node transfers between clusters as well as cluster destruction and creation during motion

Electing the Cluster Leaders

Assume each mobile node is assigned a random UID

- Each node selects the node of highest UID in its range as a cluster leader

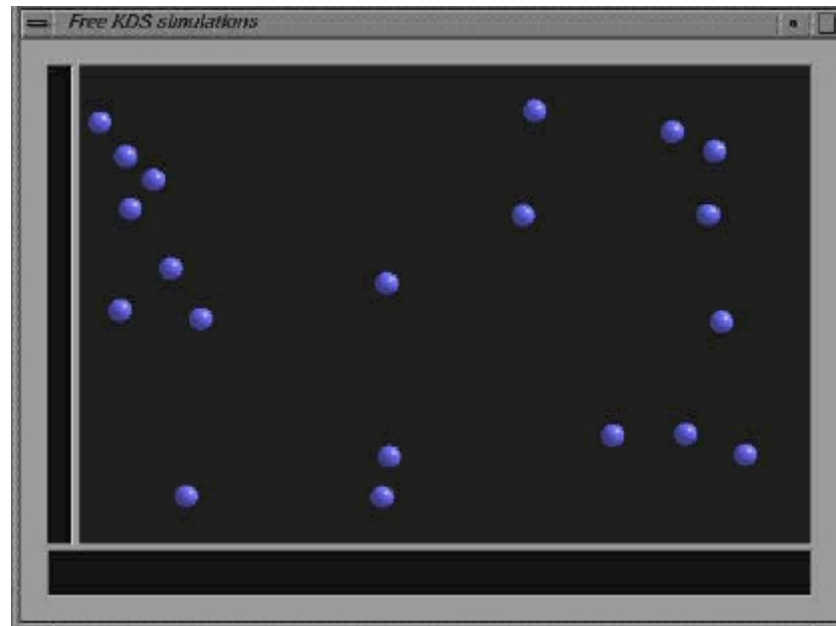
Elected leaders can be locally maintained as the nodes move. All certificates are proximity relations.

This simple-minded algorithm does not work, but a **hierarchical variant** does.

Clustering Animation



Minimum Spanning Tree Animation



Some Big Challenges for the Kinetic Approach

- Designing “smoothly varying” certificate sets
- Tailoring proofs to sensor capabilities
- Dealing with uncertainty in sensing (particle filtering, Bayes nets)
- Distributed reasoning (dKDS)

KDSs and CoSense at PARC

- The KDS reasoning machinery can focus the system sensory, computational, and communication resources to the task at hand
- Integrates well with on-going work on distributed localization and identification
- May eventually suggest new architectures for collaborative sensor systems by guiding the allocation of system resources (e.g., power) among the tasks of sensing, computation, and communication

Kinetic Data Structures are Fun

