Virtual Infrastructure for Programming Mobile Robots

Sayan Mitra
University of Illinois at Urbana-Champaign

2009 IEEE International Conference on Robotics and Automation
Formal Methods in Robotics and Automation
collaborators

- Seth Gilbert (EPFL)
- Nancy Lynch (MIT)
- Tina Nolte (MIT)
distributed mobile robotics

Robotic swarm in the lab; McLurkin (MIT)

Symbion project (EU)

Swarm monitoring environment, Belgium

500 KIVA robots deployed in Staples warehouses

Prototype mesicopter and vision of swarm exploration. Ilan Kroo (Stanford)
challenges in programming distributed robots

- key issues
  - distributed: concurrent programming is hard
  - failure prone:
    - wireless communication, delays, uncertainties
    - process failures, defects

- how to program reliable distributed (control) systems running on unreliable components?
  - two types of programs and algorithms:
    - distributed control algorithms: flocking, rendezvous, continuous consensus
    - distributed computing algorithms: atomic broadcast, replicated data, leader election
distributed control algorithms

- relatively little state is maintained
- straightforward transition rules (e.g., convex combinations)
- convergence proved under a different communication models
  - synchronous: Suzuki, Yamashita `99, Defago, Konagaya `02
  - asynchronous: Blondel, Tsitsiklis `86; Murray, Olfati-Saber, Fax `95; Cortez, Martinez, Bullo `04
  - partially synchronous: Chandy, Mitra, Pilotto `08
- typical algorithms of this class adapt to crash failures but state information (data) is lost
- other failures and joins not considered!

self-organization in nature e.g. stigmergy in bees
distributed computing algorithms

- algorithms use data-structures like stacks, queues
- algorithms are required to **terminate** or stabilize inspite of crash/byzantine failures
- impossibility of achieving consensus in asynchronous systems with crash failure: Fischer, Lynch, Paterson 1985
- techniques for fault tolerant maintenance of consistent data based on **replication**
  - ISIS (Birman et al 94), CODA, BFT (Castro, Liskov 99)

feasibility of these approaches in reliable distributed control have not been explored

Virtual Infrastructure enables us to combine both these types of algorithms
outline

- introduction
- overview of Virtual Infrastructure
- model & implementation
- self-stabilization of VI algorithm
- conclusions
a typical coordination problem

- area of interest, possibly changing over time (e.g. a forest fire, a field)
- E: set of events
  - events occur near the boundary, e.g., animal entry and exit events
- P: set of mobile robots
- goal: coordinate robots for
  - maintaining boundary
  - recording events
a typical coordination problem

- area of interest, possibly changing over time (e.g. a forest fire, a field)

- $E$: set of events
  - events occur near the boundary, e.g., animal or vehicle entry & exit

- $P$: set of mobile robots

- **goal**: coordinate robots for
  - maintaining boundary
  - recording events

- **robustness requirements**
  - must tolerate failures
  - must adapt to new robots joining
overview of Virtual Infrastructure
virtual node approach

- area partitioned into **regions**
- each region is associated with a stationary **virtual node (VN)**
  - VN is a state machine (timed automaton)
- VN programs
  - maintain relevant state
  - coordinate with neighboring VNs for accomplishing distributed control task
virtual node approach

- area partitioned into **regions**
- each region is associated with a stationary **virtual node (VN)**
  - VN: a state machine (timed automaton)
- VN programs
  - maintain relevant state
  - example: event $E_1$ occurs in region 7 then $VN_7$.count[$E_1$] ++
virtual node approach

- area partitioned into regions
- each region is associated with a stationary virtual node (VN)
  - VN: a state machine (timed automaton)
- VN programs
  - maintain relevant state
  - coordinate with neighboring VNs for accomplishing distributed control task
  - example: VN3 and VN6 interact & redistribute robots in regions 3 & 6
  - example: VN3 moves robots within region 3 for achieving equal spacing
how does it work?

- robots **emulate** VNs
  - soon after robot \( r \) enters region \( i \) it starts a VN emulation (VNE) algorithm that maintains the state of VN\(_i\)
- if multiple robots are in region \( i \), then they all maintain the state of VN\(_i\) in a consistent manner
  - replicated (timed) state machine
  - if regions are populated by robots at all times then VNs are **persistent** even as robots move around
- if a region \( i \) becomes vacant then the VN\(_i\) dies and memory is lost
key advantages

- VNs provide a programming abstraction with stationary, (relatively) **persistent virtual infrastructure**
  - VN applications are easier to program
- VNE is **self-stabilizing**
  - a robot reaches a consistent state of VN$_i$ within bounded time of entering region $i$
  - VN applications are easier to reason about
- proof of correctness of application algorithm can assume persistent, reliable infrastructure
model & implementation
physical layer

- physical nodes (robots) may fail and restart
  - \( P \): set of unique ids for nodes
  - \( v_{\text{max}} \): max speed

- nodes receive from RW (GPS)
  - location/ region information, real-time clock
  - refreshes each node at least every \( d_s \) time.

- area tiled into regions with ids in \( U \)
  - \( r \): max dist between 2 points in neighboring regions

- Pbcast local broadcast: \( bcast, brcv \)
  - non-duplicative delivery
  - bounded-time delivery: \( d_{\text{phys}} \)
  - reliable delivery within distance: \( r + d_s v_{\text{max}} \)
virtual layer

virtual layer is implemented by physical layer
- Virtual Nodes (VNs): one per regions
- Client Nodes (CNs): one per robot
  - a particular timed input/output (TIOA) automaton at a predetermined location
  - real-time clocks
  - external interface: bcast, receive, fail, restart
- Vbcast local broadcast service: vcast, vrcv
- VW: virtual world
example: virtual layer algorithm for maintaining boundary

each round:

1) each CN sends a message to its local VN, letting it know it is in the VN’s region
2) each VN exchanges messages with neighboring VNs, letting them know how many CNs it has
3) each VN calculates
   1) which of its local CNs should be assigned to other VNs
   2) what its local CNs’ new target points should be
4) VNs broadcast the new target points for CNs
5) CNs read new target points and moves to it
virtual layer

virtual layer is implemented by physical layer

- Virtual Nodes (VNs): one per regions
- Client Nodes (CNs): one per robot
  - a particular timed input/output (TIOA) automaton at a predetermined location
  - real-time clocks
  - external interface: bcast, receive, fail, restart
- Vbcast local broadcast service: vcast, vrcv
  - integrity, non-duplicative bounded-time ($\leq d$) delivery (larger than $d_{\text{phys}}$)
  - reliable delivery to all VNs & CNs in same or in neighboring region
- VW: virtual world
  - provides correct real-time to VNs
  - at least every $d_s$ time
  - can crash, restart VNs
sketch of VN Emulation algorithm

- each physical (robot) node implements
  - **TObcast**: reliable totally ordered local broadcast
  - **LE**: leader election algorithm
  - **VNE algorithm**: a round-based replicated state machine algorithm for emulating a deterministic timed automaton
- messages sent using TObcast
- maintains state of VN
  - example: robot $i$ in region $u$ detects event $E_1$ and broadcasts update to state variable $VN_u.count[E1]++$
  - only leader broadcasts on behalf of the VN
    - example: leader (e.g., robot $i$) in region $u$ broadcasts reallocation msgs to other CNs on behalf of $VN_u$
    - leader broadcasts VN state after performing VN broadcasts to maintain consistency and help joiners

Dolev et al. DISC`03, OPODIS`05
Consider a virtual layer algorithm $A$. For any execution $E$ of the emulation of $A$ in the physical layer, there exists a corresponding execution $E'$ of $A$ in the virtual layer.

$\text{amap}[A] \ || \ CW \ || \ \text{Pbcast}$ implements $A \ || \ VW \ || \ Vbcast$

TIOA $A$ implements TIOA $B$ if every externally visible behavior of $A$ is an externally visible behavior of $B$. 
self-stabilization
self-stabilization: preliminaries

- **self-stabilization**: starting from arbitrary (faulty) state $s$, the system reaches some state in a desirable set $G$
  - standard techniques exist in distributed systems literature for designing self-stabilizing algorithms [Dolev 2000]

- does not take into account the actions of the (physical) environment
- environment includes RW, Pbcast, VW, Vbcast
- we introduce a new notion of stabilization in terms of behaviors (instead of states)
stabilization

- $\alpha''$ is a **state-matched t-suffix** of execution fragment $\alpha$

\[
\begin{array}{c}
\alpha: \quad \alpha' \quad \alpha'' \\
\text{t}
\end{array}
\]

- let $B, C$ be a sets of execution fragments, $t$ be a non-negative real.
- $B$ **stabilizes** in time $t$ to $C$ if each state-matched $t$-suffix of each sequence in $B$ is a sequence in $C$.
- any behavior from $B$, has a suffix in a desirable set of behaviors $C$
self-stabilization in an environment

- **Any(A)**: automaton $A$ started in any arbitrary state
- **Reach(A)**: $A$ started in any reachable state

- A self-stabilizes in time $t$ to $L$ relative to environment $O$ if the set of executions of $\text{Any}(A) \parallel O$ stabilizes in time $t$ to set of execution fragments of $A \parallel O$ started in a legal state.
stack of self-stabilizing algorithms

- Algorithm 1 stabilizes to $L_1$ in time $T_1$
- Algorithm 2 stabilizes to $L_2$ in time $T_2$
- Algorithm 2 (starting from $L_1$) stabilizes to $L_2$ in time $T_2$
- Motion cood (MC) starting from legal state of VNE stabilizes to proportional distribution in $T_4$
- VNE algorithm (starting from $L_1 \land L_2$) stabilizes to a legal state $L$ in time $T_3$
- TObcast stabilizes to $L_1$ in time $T_1$
- Leader elec. stabilizes to $L_2$ in time $T_2$

$T_3 + \max(T_1, T_2)$

$T_4 + T_3 + \max(T_1, T_2)$
stabilization of MC

- **Theorem**: If there are no failures or recoveries of client nodes at or after some round $t_0$, then:
  - within a finite number of rounds after $t_0$, the set of CNs assigned to each region $u$ becomes fixed
  - number of CNs in a region $u$ is proportional to the percentage of the curve in $u$ within a small tolerance, and
  - all CNs in a region the curve passes through are located on the curve and are evenly spaced in the limit
conclusion

- virtual layer provides a *programming abstraction* for developing reliable distributed control
  - VNs can be used for local coordination and
  - for maintaining local state

- fault-tolerance of VN algorithms proved systematically by showing:
  - VNE is self-stabilizing + self-stabilizing VN application
    - = self-stabilizing system

- ongoing: implementation & experiments
  - on a platform of 25 desktops with simulated physical environment

- future: state-dependent sensing and distributed planning
references

- Dolev, et al. Geoquorums, DISC`03
- Dolev, et al. Virtual stationary automata, OPODIS`05
- Lynch, Mitra, Nolte, Motion coordination using virtual nodes, CDC’05
- Gilbert, Lynch, Mitra, Nolte, Self-stabilizing robot coordination over unreliable networks, SSS’08, full version to appear in ACM TAAS