Proceedings of the
44th IEEE Conference on Decision and Control, and
the European Control Conference 2005
Seville, Spain, December 12-15, 2005

Motion Coordination using Virtual Nodes

Nancy Lynch, Sayan Mitra, and Tina Nolte
CSAIL, MIT
32 Vassar Street
Cambridge, MA 02139, USA
{lynch,mitras,tnolte}@csail.mit.edu

***Abstract*— We describe how a virtual node abstraction layer can be used to coordinate the motion of real mobile nodes on a 2D plane. In particular, we consider how nodes in a mobile ad hoc network can arrange themselves along a predetermined curve in the plane, and can maintain themselves in such a configuration in the presence of changes in the underlying mobile ad hoc network, specifically, when nodes may join or leave the system or may fail. Our strategy is to allow the mobile nodes to implement a virtual layer consisting of mobile client nodes, stationary Virtual Nodes (VNs) for predetermined zones in the plane, and local broadcast communication. The VNs coordinate among themselves to distribute the client nodes between zones based on the length of the curve through those zones, while each VN directs its zone's local client nodes to move themselves to equally spaced locations on the local portion of the target curve.**

***Index Terms*— Motion coordination, virtual nodes, hybrid systems, hybrid I/O automata.**

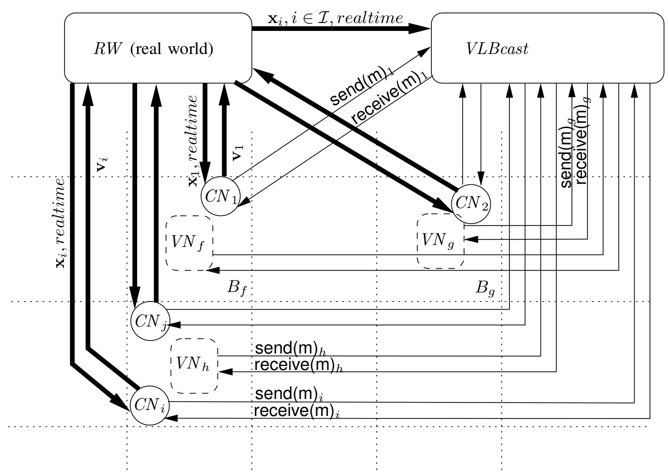

Fig. 1. Virtual Node Layer: VNs and CNs communicate using $VLBcast$.

I. Introduction

Motion coordination is the general problem of achieving some global spatial pattern of movement in a set of autonomous agents. An important motivation for studying distributed motion coordination, that is, coordination among agents with only local communication ability and therefore limited knowledge about the state of the entire system, stems from the developments in the field of mobile sensor networks. Previous work in this area includes different coordination goals, for example: flocking [8], rendezvous [1], [9], [12], deployment [2], pattern formation [14], and aggregation [7]. Owing to the intrinsic decentralized nature of sensor network applications like surveillance, search and rescue, monitoring, and exploration, centralized or leader based approaches are ruled out. However, the lack of central control makes the programming task quite difficult.

In prior work [3]–[6], we have developed a notion of "virtual nodes" for mobile ad hoc networks. A virtual node is an abstract, relatively well-behaved active node that is implemented using less well-behaved real nodes. They can be used to solve problems such as providing atomic memory [5], geographic routing [3], and point-to-point routing [4].

Here we explore the use of virtual nodes to solve motion coordination problems. Namely, we consider virtual nodes associated with predetermined, well-distributed locations in the plane, communicating among themselves and with mobile "client nodes" using local broadcast. We describe a framework for using virtual nodes to solve a simple motion coordination problem and briefly describe one way of implementing virtual nodes using the real mobile nodes. We use the Hybrid I/O Automata (HIOA) mathematical framework [11] for describing the components in our systems.

II. The Virtual Node Layer

Here we describe the virtual node layer that we will use in implementing motion coordination. The deployment space consists of a bounded square B in R^2, partitioned into a finite set of zones B_h, $h \in \mathcal{H}$. For simplicity we assume $\mathcal{B}$ is a $m \times m$ square grid, with each grid square corresponding to a zone and having sides of length b. Each boundary point of a square is unambiguously assigned to one zone. The index set $\mathcal{H}$ is the set of coordinates of the centers of all squares. For each B_h, the set $Nbrs_h$ contains the zone identifiers of the north, south, east, and west neighboring grid squares.

Our virtual layer (see Figure 1) consists of: (1) an unknown finite number of client node automata CN_i, with unique identifiers $i \in \mathcal{I}$, (2) one stationary virtual node automaton VN_h for each $h \in \mathcal{H}$, located at the center $\mathbf{o}_h$ of the square B_h, (3) a virtual communication service, $VLBcast$, for VNs and CNs, and (4) an automaton RW to model the real time and each CN's location.

*Research supported by AFRL contract number F33615-010C-1850, DARPA/AFOSR MURI contract number F49620-02-1-0325, NSF ITR contract number CCR-0121277, and DARPA-NEST contract number F33615-01-C-1896.

A mobile client node automaton CN_i, $i \in \mathcal{I}$, is an HIOA that continuously receives from RW the current time as the input variable $realtime$ and its position as the input variable $\mathbf{x}_i$, and communicates its velocity to RW through the output variable $\mathbf{v}_i$. The speed of CN_i is bounded by v_c. The trajectories of the continuous variable $\mathbf{v}_i$ and the effects of the send and receive actions are unspecified. At each point CN_i is either in active or inactive mode; we assume that, initially, finitely many nodes are active. The fail_i input action sets the mode to inactive and the $\mathsf{recover}_i$ input action sets it to active. In inactive mode, all internal and output actions are disabled, no input action except $\mathsf{recover}_i$ affects the internal or output variables, and during trajectories, the locally-controlled variables remain constant and the velocity $\mathbf{v}_i$ remains zero. We model the departure of a node from $\mathcal{B}$ as a failure. For convenience, we assume transitions are instantaneous. CN_i also has send and receive actions for interacting with the $VLBcast$ service.

A virtual node automaton VN_h, $h \in \mathcal{H}$, is an MMT automaton [13]. An MMT automaton is a special type of discrete I/O automata that has a "task" structure, which is an equivalence relation on the set of locally-controlled actions, and an upper bound parameter d_{MMT}, such that from a point in an execution where a task becomes enabled, some action in that task occurs within d_{MMT} time. VN_h can fail, disabling internal and output actions, preventing any inputs other than $\mathsf{recover}_h$ from resulting in state changes, and setting the automaton to an initial state. If a $\mathsf{recover}_h$ occurs, the VN actions become enabled with all tasks restarted. If VN_h is failed and a CN later enters B_h and remains active in the zone for d_r time, then a $\mathsf{recover}_h$ occurs within that d_r time. VN_h communicates with other VNs and CNs using the $VLBcast$ service through send_h and $\mathsf{receive}_h$ actions.

$VLBcast$ is a local broadcast service, parameterized by radius R_v and maximum message delay d_v, where $R_v \geq b$. It allows VN_h to communicate with each VN_g such that $g \in Nbrs_h$, and with CNs that are located in B_h. It does not allow CN automata to communicate with one another. This service guarantees that when any (client or virtual) node performs a $\mathsf{send}(m)_i$ action at some time t, the message is delivered within the interval $[t, t + d_v]$, by a $\mathsf{receive}(m)_h$ action, to all appropriate nodes h, that are active for the entire interval.

The RW automaton reads the velocity output $\mathbf{v}_i$ from each CN_i, $i \in \mathcal{I}$, and produces the position $\mathbf{x}_i$ and $realtime$ for CN_i and $VLBcast$.

Virtual Node Layer implementation: One implementation of this layer using mobile nodes closely follows the VMN layer implementation in [4]; mobile nodes in a zone use a replicated state machine algorithm to implement the zone's virtual node. Each mobile node runs a totally ordered broadcast service, $TOBcast$, and a Virtual Node Emulation (VNE) algorithm, for each virtual node. The $TOBcast$ service ensures that each VNE receives the same set of messages in the same order. Assuming mobile nodes are equipped with a real local broadcast service $PLBcast$, with communication radius $R_p \geq \sqrt{5}b$ and message delay d_p, $TOBcast$ is implemented using a hold strategy for received messages, where nodes don't "receive" a message until enough ($d_p + \epsilon$, ϵ small) time has passed that all other nodes in the zone will have received the message as well. Each VNE then independently maintains the state of the zone's virtual node. Whenever a VNE wishes to emulate a virtual node action, it uses $TOBcast$ to send the action suggestion to other VNEs. Once action suggestions are received, and if they are still applicable, each VNE simulates the effect of the action on its local version of the virtual node state, possibly emitting a virtual node broadcast.

The implementation provides the Virtual Node abstraction with VN task upper time bound $d_{MMT} = 2d_p + 2\epsilon$, VN-startup time $d_r = 4d_p + 5\epsilon$, and $VLBcast$ message delay $d_v = 2d_p + \epsilon$. Additional details are in the full paper [10].

III. The Motion Coordination Problem

A *differentiable parameterized curve* Γ is a differentiable map $P \rightarrow \mathcal{B}$, where the domain set P of parameter values is an interval in the real line. The curve Γ is *regular* if for every $p \in P$, $|\Gamma'(p)| \neq 0$. For $a, b \in P$, the *arc length* of a regular curve Γ from a to b, is given by $s(\Gamma, a, b) = \int_a^b |\Gamma'(p)| dp$. Γ is said to be *parameterized by arc length* if for every $p \in P$, $|\Gamma'(p)| = 1$. For a curve parameterized by arc length, $s(\Gamma, a, b) = b - a$.

For a given point $\mathbf{x} \in \mathcal{B}$, if there exists $p \in P$ such that $\Gamma(p) = \mathbf{x}$, then we say that the point $\mathbf{x}$ is on the curve Γ; abusing the notation, we write this as $\mathbf{x} \in \Gamma$. We say that Γ is a simple curve provided for every $\mathbf{x} \in \Gamma$, $\Gamma^{-1}(\mathbf{x})$ is unique. A sequence $\mathbf{x}_1, \ldots, \mathbf{x}_n$ of points in $\mathcal{B}$ are said to be *evenly spaced* on a curve Γ if there exists a sequence of parameter values $p_1 < p_2 \ldots < p_n$, such that for each i, $1 \leq i \leq n$, $\Gamma(p_i) = \mathbf{x}_i$, and for each i, $1 < i < n$, $p_i - p_{i-1} = p_{i+1} - p_i$.

In this paper we fix Γ to be a simple, differentiable curve that is parameterized by arc length. Let $P_h = \{p \in P : \Gamma(p) \in B_h\}$ be the domain of Γ in zone $B_h \subseteq \mathcal{B}$. The local part of the curve Γ in zone B_h is the restriction $\Gamma_h : P_h \rightarrow B_h$. We assume that P_h is convex for every zone $B_h \subseteq \mathcal{B}$; it may be empty for some B_h. We write $|P_h|$ for the length of the curve Γ_h. We define the *quantization* of a real number x with quantization constant $\sigma > 0$ as $q_\sigma(x) = \lceil \frac{x}{\sigma} \rceil \sigma$. For the remainder of the paper we fix σ and write q_h as an abbreviation for $q_\sigma(|P_h|)$. We write q_{min} for the minimum nonzero q_h, and q_{max} for the maximum q_h.

Our goal is to design an algorithm that runs on the physical mobile nodes such that, if there are no failures or recoveries of physical nodes after a certain point, then: (1) within finite time the set of nodes in each zone B_h, $h \in \mathcal{H}$, becomes fixed, and the size of the set is "approximately" proportional to the quantized length q_h, (2) within finite time all physical nodes in B_h for which $q_h \neq 0$ are located on Γ_h, and (3) in the limit all the nodes in each B_h are evenly spaced on Γ_h.

IV. Motion Coordination Using Virtual Nodes

The Virtual Node abstraction is used as a means to coordinate the movement of client nodes in a zone. A VN

controls the motion of the CNs in its zone by setting and broadcasting target waypoints for the CNs: VN_h, $h \in \mathcal{H}$, periodically receives information from clients in its zone, exchanges information with its neighbors, and sends out a message containing a calculated target point for each client node "assigned" to zone B_h. Informally, VN_h performs two tasks when setting the target points: (1) it re-assigns some of the CNs that are assigned to itself to neighboring VNs, and (2) it sends a target position on Γ to each CN that is assigned to itself. The objective of (1) is to prevent neighboring VNs from getting depleted of CNs and to achieve a distribution of CNs over the zones that is proportional to the length of Γ in each zone. The objective of (2) is to space the nodes evenly on Γ within each zone. A CN, in turn, receives its current position information from RW and its target location from a VN, and continuously computes a velocity vector that will take it to its latest received target point.

Each virtual node VN_h uses only information about the portions of the target curve Γ in zone B_h and neighboring zones. We assume that all client nodes know the complete curve Γ; however, we could model the client nodes in B_h as receiving inputs from another automaton about the nature of the curve in zone B_h and neighboring zones only.

A. Client Node Algorithm

The algorithm for the client node $CN(\delta)_i$, $i \in \mathcal{I}$, appears in Figure 2. The client follows a round structure, where rounds begin at times that are multiples of δ. Recall that VNs do not have access to $realtime$ whereas CN automata do. To help VNs follow the round structure, the CNs send "trigger" messages to prompt VNs to perform transitions.

At the beginning of each round, a CN sends a cn-update message to its local VN (that is, the VN in whose zone the CN currently resides). The cn-update message tells the local VN the CN's *id*, its current location in $\mathcal{B}$, and current round number.

The CN then sends an exchange-trigger message $d_v + \epsilon$ later to its local VN. An additional $d_{MMT} + 2d_v + \epsilon$ time later, the CN sends a target-trigger message to its local VN. Both these messages are trigger messages that include the CN's current location and the current round number, used by the local VN to determine whether the CN is in its zone and what the current round number is.

CN_i processes only one kind of message, target-update messages sent by its assigned VN. Each such message describes the new target location $\mathbf{x}_i^*$ for CN_i, and possibly an assignment to a different VN. CN_i continuously computes its velocity vector $\mathbf{v}_i$, based on its current position $\mathbf{x}_i$ and its target position $\mathbf{x}_i^*$, as $\mathbf{v}_i = v_c(\mathbf{x}_i - \mathbf{x}_i^*)/||\mathbf{x}_i - \mathbf{x}_i^*||$, moving it with maximum velocity towards the target.

B. Round structure

The VN_h, $h \in \mathcal{H}$, algorithm follows the CNs' round structure. However, VNs do not have access to the $realtime$ variable and must instead rely on trigger messages from CNs to determine when enough time has elapsed to perform required actions. Here we explain how we implement the round structure for a VN.

Recall that at the beginning of a round, each CN sends a cn-update message to its local VN. The CNs then send exchange-trigger messages $d_v + \epsilon$ after the beginning of the round, enough time that the cn-update messages have already been delivered, signaling to the VN that it has received all cn-update messages that were transmitted at the beginning of the round in its zone. The VN waits before using information from the cn-update messages until it receives one of the CNs' exchange-trigger messages. The VN then sends vn-update messages to its neighbors.

Each CN sends a target-trigger message to its local VN an additional $d_{MMT} + 2d_v + \epsilon$ time after it sends an exchange-trigger message. This additional time is enough

Signature:
  Input
    receive$(m)_i$, $m \in$ ({target-update} $\times\ \mathcal{B}$)
  Output
    send$(m)_i$, $m \in$ ({cn-update} $\times\ \mathcal{I} \times \mathcal{B} \times$ N)
      $\cup$ ({exchange-trigger, target-trigger} $\times\ \mathcal{B} \times$ N)
  Internal
    init$_i$

Variables:
  Input
    $\mathbf{x}_i \in \mathcal{B}$
    $realtime \in R^{\geq 0}$
  Output
    $\mathbf{v}_i \in R^2$, velocity vector
  Internal
    $\mathbf{x}^* \in \mathcal{B} \cup \{\perp\}$, target point, initially $\perp$
    round, *next-exch*, *next-target* $\in$ N $\cup \{\perp\}$, initially $\perp$

Transitions:
  Internal init$_i$
  Precondition
    round $= \perp$
  Effect
    round, *next-exch*, *next-target* $\leftarrow \lceil realtime/\delta \rceil$
    $\mathbf{x}^* \leftarrow \mathbf{x}_i$

  Input receive($\langle$target-update, *target*$\rangle)_i$
  Effect
    if *target*$(i) \neq$ *null* **then**
      $\mathbf{x}^* \leftarrow target(i)$

  Output send($\langle$cn-update, i, $\mathbf{x}_i$, *round*$\rangle)_i$
  Precondition
    realtime $=$ *round* $\cdot\ \delta$
  Effect
    round $\leftarrow$ *round* + 1

  Output send($\langle$exchange-trigger, $\mathbf{x}_i$, *next-exch*$\rangle)_i$
  Precondition
    realtime $=$ *next-exch* $\cdot\ \delta + d_v + \epsilon$
  Effect
    next-exch $\leftarrow$ *next-exch* + 1

  Output send($\langle$target-trigger, $\mathbf{x}_i$, *next-target*$\rangle)_i$
  Precondition
    realtime $=$ *next-target* $\cdot\ \delta + d_{MMT} + 3d_v + 2\epsilon$
  Effect
    next-target $\leftarrow$ *next-target* + 1

Trajectories:
  Evolve
    if $(\mathbf{x}_i = \mathbf{x}^*$ *or* $\mathbf{x}^* = \perp)$ **then** $\mathbf{v}_i = \mathbf{0}$
    else $\mathbf{v}_i = v_c \cdot (\mathbf{x}^* - \mathbf{x}_i)/||\mathbf{x}^* - \mathbf{x}||$
  Stop when
    round $= \perp$ *or realtime* $= round \cdot \delta$
      or next-exch$\cdot\delta + d_v + \epsilon$ *or next-target*$\cdot\delta + d_{MMT} + 3d_v + 2\epsilon$

Fig. 2. Client node $CN(\delta)_i$ automaton.

for all the following to have happened: (1) each neighboring VN has received an exchange-trigger message from a CN in its zone (d_v time), (2) each neighboring VN has performed a vn-update transmission to its neighboring VNs, including this one (d_{MMT} time), and (3) the neighboring VN vn-update messages have arrived (d_v time). When a VN first receives a target-trigger message for a particular round from any CN in its region, it knows it has received any vn-update messages from neighboring VNs for the round. The VN then performs some computation and transmits a target-update message to CNs local to it.

A target-update message might not be received by a CN until $d_{MMT} + 2d_v$ time after the CN sent the target-trigger message. This accounts for: (1) the time it can take for the target-trigger message to be received by the VN (d_v), (2) the time it can take for the VN to perform the target-update broadcast (d_{MMT}), and (3) the time for the broadcast to be delivered at the CN (d_v). Given the maximum distance between a point in one zone and the center of a neighboring zone, $\sqrt{2.5}b = \sqrt{(3b/2)^2 + (b/2)^2}$, and a constant speed of v_c for each client node, it can take up to $\frac{\sqrt{2.5}b}{v_c}$ time for the CN to reach its target. Also, after the CN arrives in the zone it was assigned to, up to $\sqrt{10}b/3 = \sqrt{2.5}b \cdot \frac{2}{3}$ distance from where it started, it could find the local VN is failed, and then take up to the d_r VN-startup time for it to recover.

To ensure a round is long enough for a client node to send the cn-update, exchange-trigger, and target-trigger messages, receive a target-update message, arrive at its new assigned target location, and be sure a virtual node is alive in its zone before a new round begins, we require that δ satisfy $\delta > 2d_{MMT} + 5d_v + 2\epsilon + max(\sqrt{2.5}b/v_c, \sqrt{10}b/3v_c + d_r)$.

C. VN algorithm

The algorithm for virtual node $VN(e, \rho_1, \rho_2)_h$, $h \in \mathcal{H}$, appears in Figure 3, where $e \in Z^+$ and $\rho_1, \rho_2 \in (0, 1)$ are parameters of the automaton. VN_h collects cn-update messages sent at the beginning of the round from CNs located in its zone, aggregating the location and round information from the message in a table, M. When VN_h first receives an exchange-trigger message for a particular round from any CN in its zone, VN_h tallies and computes from its table M the number of client nodes assigned to it that it has heard from in the round, and sends this information in a vn-update message to all of its neighbors.

When VH_h receives a vn-update message from a neighboring VN, it stores the CN population and round number information from the message in a table, V. When VN_h first receives a target-trigger message for a particular round from any CN in its region, VN_h uses the information in its tables M and V about the number of CNs in its zone and its neighbors' zones to calculate how many CNs assigned to itself should be reassigned and to which neighboring VNs. This is done through the assign function (see Figure 4) which calculates a partial function $assign$ mapping CN identifiers to zones that they are assigned to. If the number of CNs $y(h)$ assigned to VN_h exceeds the minimum critical number e, then assign reassigns some CNs to neighbors.

Let In_h denote the set of neighboring VNs of VN_h that are on the curve Γ and $y_h(g)$, $g \in Nbrs_h \cup \{h\}$, denote the number $num(V_h(g))$ of CNs assigned to VN_g. If $q_h \neq 0$, meaning VN_h is on the curve (lines 7–11), then we let $lower_h$ denote the subset of $Nbrs_h$ that are on the curve and have fewer assigned CNs than VN_h has after normalizing with $\frac{q_g}{q_h}$. For each $g \in lower_h$, VN_h reassigns the smaller of the following two quantities of CNs to VN_g: (1) $ra = \rho_2 \cdot [\frac{q_g}{q_h} y_h(h) - y_h(g)]/2(|lower_h| + 1)$, where $\rho_2 < 1$ is a damping factor, and (2) the remaining number of CNs over e still assigned to VN_h.

Signature:
  Input
    receive$(m)_h$, $m \in (\{$exchange-trigger, target-trigger$\} \times \mathcal{B} \times \mathsf{N}) \cup$
      $(\{$cn-update$\} \times \mathcal{I} \times \mathcal{B} \times \mathsf{N}) \cup (\{$vn-update$\} \times \mathcal{H} \times \mathsf{N} \times \mathsf{N})$
  Output
    send$(m)_h$

Constants:
  $In = \{g \in Nbrs\colon q_g \neq 0\}$

State variables:
  $M : \mathcal{I} \rightarrow \mathcal{B} \times \mathsf{N}$, partial map from CN ids to current location and round number, initially $\emptyset$. Accessors: $loc, round$.
  $V : \mathcal{H} \rightarrow \mathsf{N} \times \mathsf{N}$, partial map from VN ids to the number of CNs, and round number, initially $\{\langle g, \langle 0, 0\rangle\rangle\}$ for each $g \in Nbrs \cup \{h\}$. Accessors: $num, round$.
  send-buffer, queue of messages, initially $\emptyset$.
  vn-done, *target-done* $\in Z$, initially 0.

Derived variables:
  $locM = \lambda(i \in id(M)).\ loc(M(i))$
  $y = \lambda(g \in Nbrs \cup \{h\}).\ num(V(g))$

Transitions:
  Input receive$(\langle$cn-update, $id, loc, round\rangle)_h$
  Effect
    if $loc \in B_h$ **then**
      $M \leftarrow M \cup \{\langle id, \langle loc, round\rangle\rangle\}$

  Input receive$(\langle$exchange-trigger, $loc, round\rangle)_h$
  Effect
    if $(loc \in B_h \wedge vn\text{-}done \neq round)$ **then**
      for each $i \in id(M)$
        if $round(M(i)) \neq round$ **then**
          $M \leftarrow M \setminus \{\langle i, M(i)\rangle\}$
      $send\text{-}buffer \leftarrow send\text{-}buffer \cup \{\langle$vn-update, $h, |M|, round\rangle\}$
      $vn\text{-}done \leftarrow round$

  Input receive$(\langle$vn-update, $id, n, round\rangle)_h$
  Effect
    if $id \in Nbrs$ **then**
      $V(id) \leftarrow \langle n, round\rangle$

  Input receive$(\langle$target-trigger, $loc, round\rangle)_h$
  Effect
    if $(loc \in B_h \wedge target\text{-}done \neq round)$ **then**
      $V(h) \leftarrow \langle |M|, round\rangle$
      for each $g \in Nbrs$
        if $round(V(g)) \neq round$ **then**
          $V(g) \leftarrow \langle 0, 0\rangle$
      let $target =$ calctarget(assign$(id(M), y)$, $locM$)
        $send\text{-}buffer \leftarrow send\text{-}buffer \cup \{\langle$target-update, $target\rangle\}$
      $target\text{-}done \leftarrow round$

  Output send$(m)_h$
  Precondition
    $send\text{-}buffer \neq \emptyset \wedge m =$ **head**$(send\text{-}buffer)$
  Effect
    $send\text{-}buffer \leftarrow$ **tail**$(send\text{-}buffer)$

Tasks and bounds:
  $\{$send$(m)_h\}$, bounds $[0, d_{MMT}]$

Fig. 3. $VN(e, \rho_1, \rho_2)_h$ IOA, implementing motion coordination with parameters: safety e, damping ρ_1, ρ_2.

If $q_h = 0$, meaning VN_h is not on the curve, and VN_h has no neighbors on the curve (lines 13–17), then we let $lower_h$ denote the subset of $Nbrs_h$ with fewer assigned CNs than VN_h. For each $g \in lower_h$, VN_h reassigns the smaller of the following two quantities of CNs: (1) $ra = \rho_2 \cdot [y_h(h) - y_h(g)]/2(|lower_h| + 1)$ and (2) the remaining number of CNs over e still assigned to VN_h.

VN_h is on a *boundary* if $q_h = 0$, but there is a $g \in Nbrs_h$ with $q_g \neq 0$. In this case, $y_h(h) - e$ of VN_h's CNs are assigned equally to neighbors in In_h (lines 19–22).

The client assignments are then used to calculate new target points for local CNs through the calctarget function (see Figure 4). This function assigns to every CN_i assigned to VN_h a target point $locM_h(i) \in B_g, g \in Nbrs_h \cup \{h\}$, to move to. The target point $locM_h(i)$ is computed as follows: If CN_i is assigned to VN_g, $g \neq h$, then its target is set to the center $\mathbf{o}_g$ of B_g (lines 30–31); if CN_i is assigned to VN_h but is not located on the curve Γ_h then its target is set to the nearest point on the curve, nondeterministically choosing one if there are several (lines 32–33); if CN_i is either the first or last client node on Γ_h then its target is set to the corresponding endpoint of Γ_h (lines 35–36); if CN_i is on the curve but is not the first or last client node then its target is moved to the mid-point of the locations of the preceding and succeeding CNs on the curve (line 38). For the last two computations a sequence seq of nodes on the curve sorted by curve location is used (line 27). VN_h finally broadcasts the new target waypoints for the round through a target-update message to its CNs.

Functions:
2 **function** assign(*assignedM*: $2^{\mathcal{I}}$, *y*: *Nbrs* $\cup\{h\} \rightarrow$ N): $\mathcal{I} \rightarrow \mathcal{H}$ =
assign: $\mathcal{I} \rightarrow \mathcal{H}$, initially $\{\langle i, h\rangle\}$ for each $i \in$ *assignedM*
4 *n*: N, initially $y(h)$
ra: N, initially 0
6 **if** $y(h) > e$ **then**
if $q_h \neq 0$ **then**
8 **let** $lower = \{g \in In\text{: } \frac{q_g}{q_h} y(h) > y(g)\}$
for each $g \in lower$
10 $ra \leftarrow \mathbf{min}(\lfloor \rho_2 \cdot [\frac{q_g}{q_h} y(h) - y(g)]/2(|lower|+1) \rfloor, n - e)$
update *assign* by reassigning ra nodes from h to g
12 $n \leftarrow n - ra$
else if $In = \emptyset$ **then**
14 **let** $lower = \{g \in Nbrs : y(h) > y(g)\}$
for each $g \in lower$
16 $ra \leftarrow \mathbf{min}(\lfloor \rho_2 \cdot [y(h) - y(g)]/2(|lower|+1) \rfloor, n - e)$
update *assign* by reassigning ra nodes from h to g
18 $n \leftarrow n - ra$
else
20 $ra \leftarrow \lfloor (y(h) - e)/|In| \rfloor$
for each $g \in In$
22 update *assign* by reassigning ra nodes from h to g
return *assign*
24
function calctarget(*assign*: $\mathcal{I} \rightarrow \mathcal{H}$, *locM*: $\mathcal{I} \rightarrow \mathcal{B}$): $\mathcal{I} \rightarrow \mathcal{B}$ =
26 *seq*, indexed list of pairs in $P \times \mathcal{I}$, initially the list, for each $i \in \mathcal{I}$:
assign$(i) = h \wedge$ *locM*$(i) \in \Gamma_h$, of $\langle p, i\rangle$ where $p = \Gamma_h^{-1}(locM(i))$,
28 sorted by p, then i
for each $i \in \mathcal{I}$: *assign*$(i) \neq$ *null*
30 **if** *assign*$(i) = g \neq h$ **then**
locM$(i) \leftarrow \mathbf{o}_g$
32 **else if** *locM*$(i) \notin \Gamma_h$ **then**
locM$(i) \leftarrow$ **choose** $\{\mathbf{min}_{\mathbf{x} \in \Gamma_h}\{dist(\mathbf{x}, locM(i))\}\}$
34 **else let** $p = \Gamma_h^{-1}(locM(i))$, *seq*$(k) = \langle p, i\rangle$
if $k =$ **first**(*seq*) **then** *locM*$(i) \leftarrow \Gamma_h(\mathbf{inf}(P_h))$
36 **else if** $k =$ **last**(*seq*) **then** *locM*$(i) \leftarrow \Gamma_h(\mathbf{sup}(P_h))$
else let *seq*$(k-1) = \langle p_{k-1}, i_{k-1}\rangle$, *seq*$(k+1) = \langle p_{k+1}, i_{k+1}\rangle$
38 *locM*$(i) \leftarrow \Gamma_h(p + \rho_1 \cdot (\frac{p_{k-1}+p_{k+1}}{2} - p))$
return *locM*

Fig. 4. $VN(e, \rho_1, \rho_2)_h$ IOA functions.

V. CORRECTNESS OF ALGORITHM

We say $CN_i, i \in \mathcal{I}$, is *active* in round t if its mode is active for the duration of round t. A VN_h, $h \in \mathcal{H}$, is *active* in round t if there is some active CN_i with $\mathbf{x}_i \in B_h$ for the duration of rounds $t-1$ and t. Thus, none of the VNs is active in the starting round. We use the following notation: $In(t)$ is the set of ids $h \in \mathcal{H}$ of VNs that are active in round t and for which $q_h \neq 0$. $Out(t)$ is the set of ids $h \in \mathcal{H}$ of VNs that are active in round t and for which $q_h = 0$. $C(t)$ is the set of active CNs at round t, and $C_{in}(t)$ and $C_{out}(t)$ are the sets of active CNs located in zones with ids in $In(t)$ and $Out(t)$, respectively, at the beginning of round t.

For any pair of neighboring zones B_g and B_h, and for any round t, we use $y_g(h)(t)$ to refer to the value of $y_g(h)$ at the point in time in round t when VN_g finishes processing the first target-trigger message of round t it receives. For any $f, g \in Nbrs_h \cup \{h\}$, in the absence of failures and recoveries of CNs in round t, $y_f(h)(t) = y_g(h)(t)$; we write this simply as $y_h(t)$. We present a sequence of lemmas that together establish the following theorem:

Theorem 1: 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:
(1) the set of CNs assigned to each VN_h, $h \in \mathcal{H}$, becomes fixed, and the size of the set is proportional to the quantized length q_h within a constant additive term $\frac{10(2m-1)}{q_{min}\rho_2}$, and
(2) all client nodes in B_h for which $q_h \neq 0$ are located on Γ_h and evenly spaced on Γ_h in the limit .

Owing to shortage of space we do not give proofs for all the lemmas; they can be found in the complete version of the paper [10]. For the rest of this section we fix a particular round number t_0 and assume that no failures or recoveries of CNs occurs at or after round t_0. The first lemma states some basic facts about the assign function (see Figure 4):

Lemma 1: In every round $t \geq t_0$: (1) If $y_h(t) \geq e$ for some $h \in \mathcal{H}$, then $y_h(t+1) \geq e$, (2) $In(t) \subseteq In(t+1)$, (3) $Out(t) \subseteq Out(t+1)$, (4) $C_{in}(t) \subseteq C_{in}(t+1)$, and (5) $C_{out}(t+1) \subseteq C_{out}(t)$.

The next lemma states a key property of the assign function after round t_0: VN_g, $g \in Out(t)$, is never assigned a larger number of CNs in round $t+1$ than the largest number of CNs that were assigned to any of VN_g's neighbors in round t. Similarly, VN_g, $g \in In(t)$, never gets a density $\frac{y_g(t+1)}{q_g}$ of CNs CNs in round $t+1$ that is greater than the highest density of its neighbors in round t.

Lemma 2: In every round $t \geq t_0$, for $g, h \in \mathcal{H}$ and $h \in Nbrs_g$: (1) If $g, h \in Out(t)$, $y_h(t) = max_{f \in Nbrs_g} y_f(t)$, and $y_g(t) < y_h(t)$, then $y_g(t+1) \leq y_h(t) - 1$, and
(2) If $g, h \in In(t)$, $\frac{y_h(t)}{q_h} = max_{f \in Nbrs_g} \frac{y_f(t)}{q_f}$, and $\frac{y_g(t)}{q_g} < \frac{y_h(t)}{q_h}$, then $\frac{y_g(t+1)}{q_g} \leq \frac{y_h(t)}{q_h} - \frac{\sigma}{q_{max}^2}$.

Proof: (1) Fix g, h and t, as in the lemma statement. Since $y_h(t) > y_g(t)$ and $g, h \in Out(t)$, we see from line 16 of Figure 4 that the number of CNs that VN_g is assigned from VN_h in round t is at most $\rho_2(y_h(t) - y_g(t))/2(|lower_h(t)| + 1)$. This is at most $\rho_2(y_h(t) - y_g(t))/4$, because $y_h(t) > y_g(t)$ implies that $lower_h(t) \geq 1$. Then, the total number of CNs assigned to VN_g in round t by all four of its neighbors is at most $\rho_2(y_h(t) - y_g(t))$. Therefore, $y_g(t+1) \leq y_g(t) + \rho_2(y_h(t) - y_g(t)) = \rho_2 y_h(t) + (1 - \rho_2)y_g(t)$. As $\rho_2 < 1$, we have $y_g(t+1) < y_h(t)$. The result follows from integrality of $y_g(t+1)$ and $y_h(t)$.

(2) Similar to part (1). ■

The next lemma says there is a round T_{out} that is reached within a finite number of rounds after t_0, such that in every round $t \geq T_{out}$, the set of CNs assigned to VN_h, $h \in Out(t)$, does not change.

Lemma 3: There exists a round $T_{out} \geq t_0$ such that in any round $t \geq T_{out}$, the set of CNs assigned to VN_h, $h \in Out(t)$, is unchanged.

Proof sketch: First, we show the number of CNs assigned to VN_h, $h \in Out(t)$, remains unchanged, that is $y_h(t+1) = y_h(t)$. Let N_{out} be the total number of $h \in \mathcal{H}$ such that $q_h = 0$. For any k, $1 \leq k \leq N_{out}$, we define $max_k(t)$ to be the k^{th} largest number of CNs that are assigned to any VN_h, $h \in Out(t)$, at the beginning of round $t \geq t_0$:

$$max_k(t) \triangleq \begin{cases} max\{y_h(t) \ : h \in Out(t)\}, & \text{if } k = 1 \\ max\{y_h(t) \ : h \in Out(t) \wedge \\ \qquad y_h(t) < max_{k-1}(t)\}, & \text{otherwise.} \end{cases}$$

Let $maxvns_k(t)$ be the set of VN ids that have $max_k(t)$ CNs assigned to them. If there exists an l, $1 \leq l \leq N_{out}$, such that $\forall h \in Out(t) : max_l(t) \geq y_h(t)$, then for all k, $l < k \leq N_{out}$, $max_k(t) = 0$ and $maxvns_k(t) = \emptyset$.

Consider the function $E(t) = (|C_{out}(t)|, max_1(t), |maxvns_1(t)|, \ldots, max_{N_{out}}(t), |maxvnx_{N_{out}}(t)|)$. We show that there is a finite lower bound on the value of this function, and that for every round $t \geq t_0$, either $E(t+1) = E(t)$, that is, $t = T_{out}$, or $E(t+1)$ is less than $E(t)$ by some constant amount, meaning there is a $k, 1 \leq k \leq N_{out}$, such that for every $l, 1 \leq l < k$, the l^{th} component of $E(t+1)$ is equal to the l^{th} component of $E(t)$, and the k^{th} component of $E(t+1)$ is less than the k^{th} component of $E(t)$ by at least 1. This implies that there exists T_{out}, such that the number of CNs assigned to each VN_h, $h \in Out(t)$, $t \geq T_{out}$, remains unchanged.

Now suppose the set of CNs assigned to VN_h changes in some round $t \geq T_{out}$. Since $y_h(t+1) = y_h(t)$ for all $h \in Out(t)$, summing, $|C_{out}(t+1)| = |C_{out}(t)|$ and using Lemma 1 we get $C_{out}(t+1) = C_{out}(t)$. The only way the set of CNs assigned to VN_h could change, without changing y_h and the set C_{out}, is if there existed a cyclic sequence of VNs with ids in $Out(t)$ in which each VN gives up $c > 0$ CNs to its successor VN in the sequence, and receives c CNs from its predecessor. However, such a cycle cannot exist because the $lower$ set imposes a strict partial ordering on the VNs. ■

We fix T_{out} to be the first round after t_0, at which the property stated by Lemma 3 holds. Then, for every $t \geq T_{out}$, $In(t) = In(T_{out})$ and $C_{in}(t) = C_{in}(T_{out})$; we denote these as In and C_{in}. The next lemma states a property similar to that of Lemma 3 for VN_h in $h \in In$, and its proof is similar to the proofs of Lemma 3, and uses part (2) of Lemma 2.

Lemma 4: There exists a round $T_{stab} \geq T_{out}$ such that in every round $t \geq T_{stab}$, the set of CNs assigned to VN_h, $h \in In$, is unchanged.

We fix T_{stab} to be the first round after T_{out}, at which the property stated by Lemma 4 holds. The next lemma states that the number of CNs assigned to each VN_h, $h \in In$, in the stable assignment after T_{stab} is proportional to q_h within a constant additive term.

Lemma 5: In every round $t \geq T_{stab}$, for $g, h \in In(t)$:

$$\left| \frac{y_h(t)}{q_h} - \frac{y_g(t)}{q_g} \right| \leq \left\lceil \frac{10(2m-1)}{q_{min}\rho_2} \right\rceil.$$

Finally, to prove the second part of Theorem 1, we observe that by the beginning of round $T_{stab} + 2$, all CNs in C_{in} are located on Γ. The final piece comes form the next lemma which states that the CNs in each zone B_h, $h \in In$, are evenly spaced on Γ_h in the limit.

Lemma 6: Consider a sequence of rounds $t_1 = T_{stab}, \ldots, t_n$. As $n \to \infty$, the locations of CNs in B_h, $h \in In$, are evenly spaced on Γ_h.

References

[1] H. Ando, Y. Oasa, I. Suzuki, and M. Yamashita. Distributed memoryless point convergence algorithm for mobile robots with limited visibility. *IEEE Transactions on Robotics and Automation*, 1999.

[2] J. Cortes, S. Martinez, T. Karatas, and F. Bullo. Coverage control for mobile sensing networks. *IEEE Transactions on Robotics and Automation*, 20(2):243–255, 2004.

[3] S. Dolev, S. Gilbert, L. Lahiani, N. A. Lynch, and T. A. Nolte. Timed virtual stationary automata for mobile networks. *Technical Report MIT-LCS-TR-979a*, 2005.

[4] S. Dolev, S. Gilbert, N. A. Lynch, E. Schiller, A. A. Shvartsman, and J. L. Welch. Virtual mobile nodes for mobile ad hoc networks. In *18th International Symposium on Distributed Computing*, 2004.

[5] S. Dolev, S. Gilbert, N. A. Lynch, A. Shvartsman, and J. Welch. Geoquorums: Implementing atomic memory in mobile ad hoc networks. In *17th International Symposium on Distributed Computing*, 2003.

[6] S. Dolev, S. Gilbert, N. A. Lynch, A. Shvartsman, and J. Welch. Geoquorums: Implementing atomic memory in mobile ad hoc networks. *Technical Report MIT-LCS-TR-900*, 2003.

[7] V. Gazi and K. M. Passino. Stability analysis of swarms. *IEEE Transactions on Automatic Control*, 48(4):692–697, 2003.

[8] A. Jadbabaie, J. Lin, and A. S. Morse. Coordination of groups of mobile autonomous agents using nearest neighbor rules. *IEEE Transactions on Automatic Control*, 48(6):988–1001, 2003.

[9] J. Lin, A. S. Morse, and B. Anderson. Multi-agent rendezvous problem. In *42nd IEEE Conference on Decision and Control*, 2003.

[10] N. A. Lynch, S. Mitra, and T. Nolte. Motion coordination using virtual nodes. *Technical Report MIT-LCS-TR-986*, 2005.

[11] N. A. Lynch, R. Segala, and F. Vaandrager. Hybrid I/O automata. *Information and Computation*, 185(1):105–157, August 2003.

[12] S. Martinez, J. Cortes, and F. Bullo. On robust rendezvous for mobile autonomous agents. In *IFAC World Congress*, 2005.

[13] M. Merritt, F. Modugno, and M. Tuttle. Time constrained automata. In *2nd International Conference on Concurrency Theory*, 1991.

[14] I. Suzuki and M. Yamashita. Distributed autonomous mobile robots: Formation of geometric patterns. *SIAM Journal of computing*, 28(4):1347–1363, 1999.