Self-Stabilizing Mobile Robot Formations with Virtual Nodes

Seth Gilbert[1], Nancy Lynch[2], Sayan Mitra[3⋆], and Tina Nolte[2]

[1] Ecole Polytechnique Fédérale, Lausanne
[2] Massachusetts Institute of Technology
[3] University of Illinois at Urbana-Champaign

Abstract. In this paper, we describe how virtual infrastructure can be used to coordinate the motion of mobile robots in a 2-dimensional plane in the presence of dynamic changes in the underlying mobile ad hoc network, i.e., nodes joining, leaving, or failing. The mobile robots cooperate to implement a VSA Layer, in which a virtual stationary automaton (VSA) is associated with each region of the plane. The VSAs coordinate among themselves to distribute the robots as needed throughout the plane. The resulting motion coordination protocol is self-stabilizing, in that each robot can begin the execution in any arbitrary state and at any arbitrary location in the plane. In addition, self-stabilization ensures that the robots can adapt to changes in the desired formation.

1 Introduction

We study the problem of coordinating autonomous mobile devices. Consider, for example, firefighting robots deployed in forests and other fire-prone wilderness areas. Significant levels of coordination are required in order to combat the fire: the fire should be surrounded, "firebreaks" should be created, and it should be doused with water; in additiona, the firefighters may need to direct the actions of (potentially autonomous) helicopters carrying water. Similar scenarios arise in a variety of contexts, including search and rescue, emergency disaster response, remote surveillance, and military engagement, among many others. In fact, autonomous coordination has long been a central problem in mobile robotics.

We focus on a generic coordination problem that captures many of the complexities associated with these real-world scenarios. We assume that the mobile robots are deployed in a large two-dimensional plane, and that they can coordinate via local communication using wireless radios. The robots must arrange themselves to form a particular pattern, specifically, they must spread themselves evenly along a continuous curve drawn in the plane. In the firefighting example described above, this curve might form the perimeter of the fire.

These types of coordination problems can be quite challenging due to the dynamic and unpredictable environment that is inherent to wireless ad hoc networks. Robots may be continuously joining and leaving the system, and they

⋆ Supported by NSF CSR program (Embedded & Hybrid systems area) under grant NSF CNS-0614993.

may fail. In addition, wireless communication is notoriously unreliable due to collisions, contention, and various wireless interference.

Virtual infrastructure has been proposed as a new tool for building reliable and robust applications in unreliable wireless ad hoc networks (e.g., [1,2,3,4]). The basic principle motivating virtual infrastructure is that many of the challenges in a dynamic networks could be avoided if there were real network infrastructure available. Unfortunately, in many contexts, such infrastructure is unavailable. Thus, the virtual infrastructure abstraction emulates real infrastructure in ad hoc networks. It has already been observed that virtual infrastructure simplifies several important problems, including distributed shared memory [2], tracking mobile devices [5], and geographic routing [1].

In this paper, we rely on a virtual infrastructure known as the Virtual Stationary Automata Layer (VSA Layer) [6,7]. Each robot is modelled as a *client* which interacts with *virtual stationary automata* (VSAs) via a (virtual) communication service. VSAs are distributed throughout the world, each assigned permanently to its own region. An advantage of VSAs is that they are less likely to fail than an individual mobile robot. Notice that the VSAs do not actually exist in the real world; they are emulated by the underlying mobile robots.

The VSA Layer is modeled in the Timed Input/Output Automata (TIOA) [8] framework. In TIOA parlance, an *emulation* is an implementation relationship between two sets of TIOAs: those that specify the VSA Layer and those that implement it. The emulation transforms an algorithm designed for the VSA Layer into an algorithm that runs directly on the mobile robots. An execution resulting from this transformation looks as if the original program is running on the VSA Layer; formally, the traces of the transformed system, restricted to non-broadcast actions at the client nodes, are traces of the VSA Layer. In [6,7], we show how to emulate the VSA Layer in a wireless network of mobile robots.

Here, we show how to use the VSA Layer to implement a reliable and robust protocol for coordinating mobile robots. The protocol relies on the VSAs to peform the coordination. Each VSA decides based on its local information which robots to keep in its own region and which to assign to neighboring regions. For each robot that remains, the VSA determines where the robot should go. In order that the robot coordination be robust, our coordination protocol is *self-stabilizing*, meaning that each robot can begin in an arbitrary state, in an arbitrary location in the network, and yet the distribution of the robots will converge to the specified curve. When combined with a self-stabilizing implementation of the VSA Layer, as is presented in [6,7], we end up with an entirely self-stabilizing solution for the problem of autonomous robot coordination.

Self-stabilization provides many advantages. Given the unreliable nature of wireless networks, occasionally (due to aberrant interference) messages may be lost, disrupting the protocol; a self-stabilizing algorithm can readily recover from this. In addition, a self-stabilizing algorithm can cope with more dynamic coordination problems when the desired formation of robots may change. In the firefighting example above the formation of firefighting robots must adapt as the fire evolves. A self-stabilizing algorithm can easily adapt to these changes.

The remainder of this paper is organized as follows. First, in Section 2, we discuss some of the related work. Next, in Section 3, we discuss the VSA Layer model. In Section 4 we describe the motion coordination problem, and describe our algorithm that solves it. In Section 5, we show that the algorithm is correct, and in Section 6, we show that the algorithm is self-stabilizing.

2 Related Work

The problem of motion coordination has been studied in a variety of contexts, including: flocking [9]; rendezvous [10,11,12]; aggregation [13]; deployment and regional coverage [14]; and pattern formation [15]. Control theory literature contains several algorithms for achieving spatial patterns [16,17,18,19]. These assume that agents process information and communicate reliably and synchronously.

Asynchronous vision-based model have also been investigated in [15,20,21,22] and [23]. In this model, agents are asynchronous, oblivious, and anonymous. Each agent repeatedly performs *look, compute*, and *move* actions to compute its next target position based on the current position of other visible agents. The class of patterns that can be formed depends on the common knowledge of the agents, such as common compass and common coordinates [15,23].

We have previously presented a protocol for coordinating mobile devices using virtual infrastructure in [24]. This earlier protocol relies on a more powerful class of virtual infrastructure (see [6,7]), and hence, our new protocol is somewhat simpler (and more elegant). Moreover, the new protocol is self-stabilizing, which allows both for better fault-tolerance, and also the ability to tolerate dynamic changes in the desired pattern of motion. Virtual infrastructure has also been considered in [25] in the context of coordinating airplane flight.

3 Virtual Stationary Automata

The Virtual Stationary Automata (VSA) infrastructure has been presented earlier in [6,7]. The architecture of this abstraction layer is shown in Figure 1. In this section, we informally describe these components.

Network tiling. We fix R to be a closed, bounded and connected subset of $\mathbb{R}^2$, and U, P to be two totally ordered index sets. R models the physical space in which the robots reside; we call it the *deployment space*. U and P serve as the index sets for regions in R and for the participating robots, respectively. A network tiling divides R into a set of *regions* $\{R_u\}_{u \in U}$, such that: (i) for each $u \in U$, R_u is a closed, connected subset of R, and (ii) for any $u, v \in U$, R_u and R_v may overlap only at their boundaries. For any $u, v \in U$, the corresponding regions are said to be *neighbors* if $R_u \cap R_v \neq \emptyset$. This neighborhood relation, $nbrs$, induces a graph on the set of regions. We assume that the resulting graph is connected. Throughout this paper, we assume that each region has at most four neighbors; generalizing to an arbitrary number of neighbors is straightforward. We define

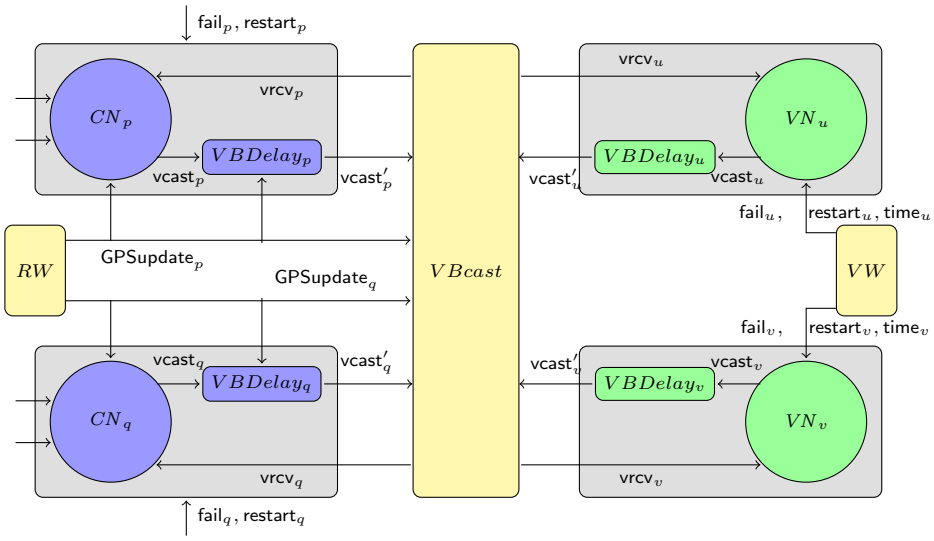

Fig. 1. Virtual Stationary Automata layer.

the distance between regions u and v, denoted $regDist(u, v)$, as the minimum number of hops between u and v in the graph. The diameter of the graph, i.e., the distance between the farthest regions, is denoted by D, and the largest Euclidean distance between any two points in any region is denoted by r.

Real World (RW) Automaton. RW is an external source of occasional but reliable time and location information for participating robots. The RW automaton is parameterized by: (a) $v_{max} > 0$, a maximum speed, and (b) $\epsilon_{sample} > 0$, a maximum time gap between successive updates for each robot. The RW automaton maintains three key variables: (a) a continuous variable now representing true system time; now increases monotonically at the same rate as real-time starting from 0. (b) An array $vel[P \to R \cup \{\perp\}]$; for $p \in P$, $vel(p)$ represents the current velocity of robot p. Initially $vel(p)$ is set to $\perp$, and it is updated by the robots when their velocity changes. (c) an array $loc[P \to R]$; for $p \in P$, $loc(p)$ represents the current location of robot p. Over any interval of time, robot p may move arbitrarily in R provided its path is continuous and its maximum speed is bounded by v_{max}. Automaton RW performs the $\mathsf{GPSupdate}(l, t)_p$ action, $l \in R, t \in \mathbb{R}_{\geq 0}, p \in P$, to inform robot p about its current location and time. For each p, some $\mathsf{GPSupdate}(,)_p$ action must occur every ϵ_{sample} time.

Virtual World (VW) Automaton. VW is an external source of occasional but reliable time information for VSAs. Similar to RW's $\mathsf{GPSupdate}$ action for clients, VW performs $\mathsf{time}(t)_u$ output actions notifying VSAs of the current time. One such action occurs at time 0, and they are repeated at least every ϵ_{sample} time thereafter. Also, VW nondeterministically issues fail_u and $\mathsf{restart}_u$ outputs for each $u \in U$, modelling the fact that VSAs may fail and restart.

Mobile client nodes. For each $p \in P$, the mobile client node CN_p is a TIOA modeling the client-side program executed by the robot with identifier p. CN_p has a local clock variable, *clock* that progresses at the rate of real-time, and is initially $\perp$. CN_p may have arbitrary local non-*failed* variables. Its external interface at least includes the GPSupdate inputs, $\mathsf{vcast}(m)_p$ outputs, and $\mathsf{vrcv}(m)_p$ inputs. CN_p may have additional arbitrary non-fail and non-restart actions.

Virtual Stationary Automata (VSAs). A VSA is a deterministic clock-equipped abstract virtual machine. For each $u \in U$, there is a corresponding VSA VN_u which is associated with the geographic region R_u. VN_u has a local clock variable *clock* which progresses at the rate of real-time. (It is initially $\perp$ before the first time input.) VN_u has the following external interface: (a) **Input** $\mathsf{time}(t)_u, t \in \mathbb{R}^{\geq 0}$, models an update at time t; it sets node VN_u's *clock* to t. (b) **Output** $\mathsf{vcast}(m)_u, m \in Msg$, models VN_u broadcasting message m. (c) **Input** $\mathsf{vrcv}(m)_u, m \in Msg$, models VN_u receiving a message m. VN_u may have additional non-*failed* variables and non-fail and non-restart internal actions.

VBDelay *Automata.* Each client and VSA node is associated with a *VBDelay* buffer that delays messages when they are broadcast for up to e time. This buffer takes as input a $\mathsf{vcast}(m)$ from the node and relays the message to the *VBcast* service after some delay of at most e. In the case of VSA nodes, there is no delay.

VBcast *Automaton.* Each client and virtual node has access to the virtual broadcast communication service *VBcast*. The service is parameterized by a constant $d > 0$ which bounds message delays. *VBcast* takes each $\mathsf{vcast}'(m, f)_i$ input (from the delay buffers) and delivers the message m via $\mathsf{vrcv}(m)$ at each client or virtual node that is in the same region as the initial sender, when the message was first sent, along with those in neighboring regions. The *VBcast* service guarantees that in each execution of *VBcast* there is a correspondence between $\mathsf{vrcv}(m)$ actions and $\mathsf{vcast}'(m, f)_i$ actions such that: (i) each vrcv occurs *after and within* d *time* of the corresponding vcast', (ii) at most one vrcv at a process is mapped to each vcast'. (iii) a message originating from some region u must be received by all robots that are in R_u or its neighbors throughout the transmission period.

A VSA layer *algorithm* is an assignment of a TIOA program to each client and VSA. We denote the set of all V-algorithms is as $VAlgs$. We now define a *VLayer*, i.e., a VSA layer with failure-prone clients and VSAs.

Definition 1. *Let* alg *be an element of* $Valgs$. $VLNodes[alg]$, *the* fail-transformed *nodes of the VSA layer parameterized by* alg, *is the composition of each* $alg(i)$, *modified so as to fail by crashing, with a* VBDelay *buffer, for all* $i \in P \cup U$. $VLayer[alg]$, *the* VSA layer parameterized by alg, *is the composition of* $VLNodes[alg]$ *with* $RW \| VW \| VBcast$.

4 Motion Coordination using Virtual Nodes

In this paper we fix $\Gamma : A \to R$ to be a simple, differentiable curve on R that is parameterized by arc length. The domain set A of parameter values is an interval

in the real line. We also fix a particular network tiling given by the collection of regions $\{R_u\}_{u \in U}$ such that each point in Γ is also in some region R_u. Let $A_u \triangleq \{p \in A : region(\Gamma(p)) = u\}$ be the domain of Γ in region u. We assume that A_u is convex for every region u; it may be empty for some u. The local part of the curve Γ in region u is the restriction $\Gamma_u : A_u \to R_u$. We write $|A_u|$ for the length of the curve Γ_u. We define the *quantization* of a real number x with quantization constant $\sigma > 0$ as $q_\sigma(x) = \lceil \frac{x}{\sigma} \rceil \sigma$. We fix σ, and write q_u as an abbreviation for $q_\sigma(|A_u|)$, q_{min} for the minimum nonzero q_u, and q_{max} for the maximum q_u.

Our goal is to design an algorithm for mobile robots such that, once the failures and recoveries cease, within finite time all the robots are located on Γ and as time progresses they eventually become equally spaced on Γ. Formally, if no fail and restart actions occur after time t_0, then:

(1) there exists a constant T, such that for each $u \in U$, within time $t_o + T$ the set of robots located in R_u becomes fixed and its cardinality is roughly proportional to q_u; moreover, if $q_u \neq 0$ then the robots in R_u are located on[4] Γ_u, and
(2) as time goes to infinity, all robots in R_u are evenly spaced[5] on Γ_u.

4.1 Solution Using Virtual Node Layer

The VSA Layer is used as a means to coordinate the movement of client nodes, i.e., robots. A VSA controls the motion of the clients in its region by setting and broadcasting target waypoints for the clients: VSA VN_u, $u \in U$, periodically receives information from clients in its region, exchanges information with its neighbors, and sends out a message containing a calculated target point for each client node "assigned" to region u. VN_u performs two tasks when setting the target points: (1) it re-assigns some of the clients that are assigned to itself to neighboring VSAs, and (2) it sends a target position on Γ to each client that is assigned to itself. The objective of (1) is to prevent neighboring VSAs from getting depleted of robots and to achieve a distribution of robots over the regions that is proportional to the length of Γ in each region. The objective of (2) is to space the nodes evenly on Γ within each region. The client algorithm, in turn, receives its current position information from RW and computes a velocity vector for reaching its latest received target point from a VSA.

Each virtual node VN_u uses only information about the portions of the target curve Γ in region u and neighboring regions. We assume that all client nodes know the complete curve Γ; however, we could model the client nodes in u as receiving external information about the nature of the curve in region u and neighboring regions only.

[4] For a given point $\mathbf{x} \in R$, if there exists $p \in A$ 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 $\mathbf{x} \in \Gamma$.

[5] A sequence $\mathbf{x}_1, \ldots, \mathbf{x}_n$ of points in R is 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, $1 < i < n$, $p_i - p_{i-1} = p_{i+1} - p_i$.

Signature:
  Input $\mathsf{time}(t)_u$, $t \in \mathbb{R}^{\geq 0}$
  Input $\mathsf{vrcv}(m)_u$, $m \in (\{\mathsf{cn\text{-}update}\} \times P \times R) \cup (\{\mathsf{vn\text{-}update}\} \times U \times \mathbb{N})$
  Output $\mathsf{vcast}(m)_u$, $m \in (\{\mathsf{vn\text{-}update}\} \times \{u\} \times \mathbb{N}) \cup (\{\mathsf{target\text{-}update}\} \times (P \to R))$

State:
  analog $clock$: $\mathbb{R}^{\geq 0} \cup \{\perp\}$, initially $\perp$.
  M:$P \to R$, initially $\emptyset$.
  $V : U \to \mathbb{N}$, initially $\emptyset$.

Trajectories:
  evolve if $clock \neq t$ **then** $\mathbf{d}(clock) = 1$ **else** $\mathbf{d}(clock) = 0$
  stop when Any precondition is satisfied.

Transitions:
  Input $\mathsf{time}(t)_u$
  Effect: **if** $clock \neq t \vee t$ **mod** $\delta \notin (0,\ e + 2d + 2\epsilon\,]$ **then** $M,\ V \leftarrow \emptyset$; $clock \leftarrow t$

  Input: $\mathsf{vrcv}(\langle \mathsf{cn\text{-}update},\ id,\ loc\rangle)_u$
  Effect: **if** $u = region(loc)$ **and** $clock$ **mod** $\delta \in (0,\ d\,]$ **then** $M(id) \leftarrow loc$; $V \leftarrow \emptyset$

  Output: $\mathsf{vcast}(\langle \mathsf{vn\text{-}update},\ u,\ n\rangle)_u$
  Precondition: $(clock$ **mod** $\delta) = d{+}\epsilon$ **and** $n{=}\ |M|{\neq}\ 0$ **and** $V{\neq}\ \{\langle u,\ n\rangle\}$
  Effect: $V \leftarrow \{\langle u,\ n\rangle\}$

  Input $\mathsf{vrcv}(\langle \mathsf{vn\text{-}update},\ id,\ n\rangle)_u$
  Effect: **if** $id \in nbrs(u)$ **then** $V(id) \leftarrow n$

  Output $\mathsf{vcast}(\langle \mathsf{target\text{-}update},\ target\rangle)_u$
  Precondition: $(clock$ **mod** $\delta) = e + 2d + 2\epsilon$ **and** $M \neq \emptyset$
  $target = \mathsf{calctarget}(\mathsf{assign}(id(M),\ V),\ M)$
  Effect: $M,\ V \leftarrow \emptyset$

Fig. 2. TIOA $VN(k, \rho_1, \rho_2)_u$ with parameters: safety k; damping ρ_1, ρ_2.

4.2 Client Node Algorithm (*CN*)

The algorithm for the client node $CN(\delta)_p$, $p \in P$ follows a round structure, where rounds begin at times that are multiples of δ. At the beginning of each round, a *CN* stops moving and sends a cn-update message to its local VSA (that is, the VSA in whose region the *CN* currently resides). The cn-update message tells the local VSA the *CN*'s *id* and its current location in R. The local VN then sends a response to the client, i.e., a target-update message. Each such message describes the new target location $\mathbf{x}_p^*$ for CN_p, and possibly an assignment to a different region. CN_p computes its velocity vector $\mathbf{v}_p$, based on its current position $\mathbf{x}_p$ and its target position $\mathbf{x}_p^*$, as $\mathbf{v}_p = (\mathbf{x}_p - \mathbf{x}_p^*)/||\mathbf{x}_p - \mathbf{x}_p^*||$ and communicates $v_{max}\mathbf{v}_p$ to RW, moving it with maximum velocity towards the target.

4.3 Virtual Stationary Node Algorithm (*VN*)

The algorithm for virtual node $VN(k, \rho_1, \rho_2)_u$, $u \in U$, appears in Figure 2, where $k \in \mathbb{Z}^+$ and $\rho_1, \rho_2 \in (0, 1)$ are parameters of the TIOA. VN_u collects cn-update messages sent at the beginning of the round from *CN*'s located in region R_u, and aggregates the location and round information in a table, M. When $d + \epsilon$ time

```
1  function assign(assignedM: 2^P, y: nbrs^+(u) → ℕ) =
2    assign: P → U, initially {⟨i, u⟩} for each i ∈ assignedM
3    n: ℕ, initially y(u); ra: ℕ, initially 0
4    if y(u) > k then
5      if q_u ≠ 0 then
6        let lower = {g ∈ nbrs(u): (q_g/q_u) y(u) > y(g)}
7        for each g ∈ lower
8          ra ← min(⌊ρ2 · ⌊(q_g/q_u) y(u) − y(g)⌋/2(|lower|+1)⌋, n − k)
9          update assign by reassigning ra nodes from u to g
10         n ← n − ra
11     else if {v ∈ nbrs(u): q_v ≠ 0} = ∅ then
12       let lower = {g ∈ nbrs(u) : y(u) > y(g)}
13       for each g ∈ lower
14         ra ← min(⌊ρ2 · [y(u) − y(g)]/2(|lower|+1)⌋, n − k)
15         update assign by reassigning ra nodes from u to g
16         n ← n − ra
17     else  ra ← ⌊ (y(u) -k)/ |{v ∈ nbrs(u): q_v ≠ 0}| ⌋
18       for each g ∈ {v ∈ nbrs(u): q_v ≠ 0}
19         update assign by reassigning ra nodes from u to g
20   return assign
21
22 function calctarget(assign: P → U, locM: P → R) =
23   seq: indexed list of pairs in A × P, sorted by the index A and then , P, initially the list:
24     ⟨p, i⟩, ∀i ∈ P : (assign(i)= u) and (locM(i) ∈ Γ_u) and p= Γ_u^{-1}(locM(i))
25   for each i ∈ P : assign(i) ≠ null
26     if assign(i) = g ≠ u then locM(i) ← o_g
27     else if locM(i) ∉ Γ_u then locM(i) ← choose {min_{x∈Γ_u}{dist(x, locM(i))}}
28     else let p = Γ_u^{-1}(locM(i)), seq(k) = ⟨p, i⟩
29       if k = first(seq) then locM(i) ← Γ_u(inf(A_u))
30       else if k = last(seq) then locM(i) ← Γ_u(sup(A_u))
31       else let seq(k − 1) = ⟨p_{k−1}, i_{k−1}⟩
32         seq(k + 1) = ⟨p_{k+1}, i_{k+1}⟩
33         locM(i) ← Γ_u(p + ρ_1 · ((p_{k−1}+p_{k+1})/2 − p))
34   return locM
```

Fig. 3. Functions assign and calctarget for the case where $VN(k, \rho_1, \rho_2)_u$ has at most 4 neighbors.

passes from the beginning of the round, VN_u computes from 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 VN_u receives a vn-update message from a neighboring VN, it stores the CN population information in a table, V. When $e + d + \epsilon$ time from the sending of its own vn-update passes, VN_u uses the information in its tables M and V about the number of CNs in its and its neighbors' regions to calculate how many CNs assigned to itself should be reassigned and to which neighbor. This is done through the assign function, and these assignments are then used to calculate new target points for local CNs through the calctarget function (see Figure 3).

If the number of CNs assigned to VN_u exceeds the minimum *safe number* k, then assign reassigns some CNs to neighbors. Let In_u denote the set of neighboring VNs of VN_u that are on the curve Γ and $y_u(g)$, denote the number $num(V_u(g))$ of CNs assigned to VN_g, where g is either u or a neigh-

bor of u. If $q_u \neq 0$, meaning VN_u is on the curve then we let $lower_u$ denote the subset of $nbrs(u)$ that are on the curve and have fewer assigned CNs than VN_u has after normalizing with $\frac{q_g}{q_u}$. For each $g \in lower_u$, VN_u reassigns the smaller of the following two quantities of CNs to VN_g: (1) $ra = \rho_2 \cdot [\frac{q_g}{q_u} y_u(u) - y_u(g)]/2(|lower_u| + 1)$, where $\rho_2 < 1$ is a *damping factor*, and (2) the remaining number of CNs over k still assigned to VN_u.

If $q_u = 0$, meaning VN_u is not on the curve, and VN_u has no neighbors on the curve (lines 11–15), then we let $lower_u$ denote the subset of $nbrs(u)$ with fewer assigned CNs than VN_u. For each $g \in lower_u$, VN_u reassigns the smaller of the following two quantities of CNs: (1) $ra = \rho_2 \cdot [y_u(u) - y_u(g)]/2(|lower_u|+1)$ and (2) the remaining number of CNs over k still assigned to VN_u. VN_u is on a *boundary* if $q_u = 0$, but there is a $g \in nbrs(u)$ with $q_g \neq 0$. In this case, $y_u(u) - k$ of VN_u's CNs are assigned equally to neighbors in In_u (lines 17–19).

The calctarget function assigns to every CN_p in the region of VN_u a target point $locM_u(p)$, either in region u or one of u's neighbors. The target point $locM_u(p)$ is computed as follows: If CN_p is assigned to VN_g, $g \neq u$, then its target is set to the center $\mathbf{o}_g$ of region g (lines 26–26); if CN_p is assigned to VN_u but is not located on the curve Γ_u then its target is set to the nearest point on the curve, nondeterministically choosing one (lines 27–27); if CN_p is either the first or last client node on Γ_u then its target is set to the corresponding endpoint of Γ_u (lines 29–30); if CN_p 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 33). For the last two computations a sequence *seq* of nodes on the curve sorted by curve location is used (line 24). Lastly, VN_u broadcasts new waypoints via a target-update message to its clients.

Round length. Let r be the maximum Euclidean distance between points in neighboring regions. It can take $\frac{r}{v_{max}}$ time for a client to reach its target. After the client arrives, the VN may have failed. Let d_r be the time it takes a VN to restart. During each round: a client sends a cn-update, the VNs exchange information, clients receive target-updates, clients move to their new target and restart any VNs. This requires that δ satisfy $\delta > 2e + 3d + 2\epsilon + r/v_{max} + d_r$.

5 Correctness of Algorithm

In this section we describe the steps in proving Theorem 1; the complete proofs will appear in the full version of the paper. We define round t as the interval of time $[\delta(t-1), \delta \cdot t)$. That is, round t begins at time $\delta(t-1)$ and is completed by time $\delta \cdot t$. We say $CN_p, p \in P$, is *active* in round t if node p is not failed throughout round t. A VN_u, $u \in U$, is *active* in round t if there is some active CN_p such that $region(\mathbf{x}_p) = u$ for the duration of rounds $t-1$ and t. Thus, by definition, none of the VN s is active in the first round.

Let $In(t) \subseteq VN$ denote the identifiers $u \in U$ such that VN_u is active in round t and $q_u \neq 0$. The set $Out(t) \subseteq VN$ denote the identifiers $u \in U$ such that VN_u is active in round t and $q_u = 0$. The set $C(t)$ is the subset of active CNs

at round t, and $C_{in}(t)$ and $C_{out}(t)$ are the sets of active CNs located in regions with ids in $In(t)$ and $Out(t)$, respectively, at the beginning of round t.

For every pair of regions u, w and for every round t, we define $y(w,t)_u$ to be the value of $V(w)_u$ (i.e., the number of clients u believes are available in region w) immediately prior to VN_u performing a vcast_u in round t. If there are no new client failures or recoveries in round t, then for every pair of regions $u, w \in nbrs^+(v)$, we can conclude that $y(v,t)_u = y(v,t)_w$, which we denote simply as $y(v,t)$. We define $\rho_3 \triangleq \frac{q_{max}^2}{(1-\rho_2)\sigma}$.

For the rest of this section we fix a particular round number t_0 and assume that $\forall p \in P$, no fail_p or $\mathsf{recover}_p$ events occur at or after round t_0. First we establish that in every round $t \geq t_0$: (1) If $y(u,t) \geq k$ for some $u \in U$, then $y(u,t+1) \geq k$; (2) $In(t) \subseteq In(t+1)$; (3) $Out(t) \subseteq Out(t+1)$. Next, we identify a round $t_1 \geq t_0$ after which the set of regions $In(t)$ and $Out(t)$ remain fixed. That is, we show that there exists a round $t_1 \geq t_0$ such that for every round $t \in [t_1, t_1 + (1+\rho_3)m^2n^2]$: (1) $In(t) = In(t_1)$; (2) $Out(t) = Out(t_1)$; (3) $C_{in}(t) \subseteq C_{in}(t+1)$; and (4) $C_{out}(t+1) \subseteq C_{out}(t)$. We fix t_1 such that it satisfies the above conditions. The next lemma states that eventually, regions bordering on the curve stop assigning clients to regions that are on the curve.

Lemma 1. *There exists some round $t_2 \in [t_1, t_1 + (1+\rho_3)m^2n^2]$ such that for every round $t \in [t_2, t_2 + (1+\rho_3)m^2n]$: if $u \in Out(t)$ and $v \in In(t)$ and if u and v are neighboring regions, then u does not assign any clients to v in round t.*

Fix t_2 for the rest of this section such that it satisfies Lemma 1. From the above discussion, it follows that in every round $t \geq t_1$, $In(t) = In(t_1)$ and $Out(t) = Out(t_1)$; we denote these simply as In and Out. The next lemma states a key property of the assign function after round t_1. For a round $t \geq t_1$, consider some VN_u, $u \in Out(t)$, and assume that VN_w is the neighbor of VN_u assigned the most clients in round t. Then we can conclude that VN_u is assigned no more clients in round $t+1$ than VN_w is assigned in round t. A similar claim holds for regions in $In(t)$, but in this case with respect to the *density* of clients with respect to the quantized length of the curve. The next lemma states that there exists a round T_{out} such that in every round $t \geq T_{out}$, the set of CNs assigned to region $u \in Out(t)$ does not change.

Lemma 2. *There exists a round $T_{out} \in [t_2, t_2 + m^2n$ such that in any round $t \geq T_{out}$, the set of CNs assigned to VN_u, $u \in Out(t)$, is unchanged.*

For the rest of the section we fix T_{out} to be the first round after t_0, at which the property stated by Lemma 2 holds. This implies that in every round $t \geq T_{out}$, $C_{In}(t) = C_{In}(t_1)$ and $C_{Out}(t) = C_{Out}(t_1)$; we denote these simply as C_{In} and C_{Out}. The next lemma states a property similar to that of Lemma 2 for VN_u, $u \in In$, and the argument is similar to the proof of Lemma 2.

Lemma 3. *There exists a round $T_{stab} \in [T_{out}, T_{out} + \rho_3 m^2 n]$ such that in every round $t \geq T_{stab}$, the set of CNs assigned to VN_u, $u \in In$, is unchanged.*

We prove that the number of clients located in regions with ids in Out is upper-bounded by $O(m^3)$. Next, fixing T_{stab} to be the first round after T_{out} at which the property stated by Lemma 3 holds, we are able to prove that the number of clients assigned to each VN_u, $u \in In$, in the stable assignment after T_{stab} is proportional to q_u within a constant additive term. From line 27 of Figure 3, it follows that by the beginning of round $T_{stab} + 2$, all CNs in C_{in} are located on the curve Γ, satisfying our first goal. The next lemma states that the locations of the CNs in each region $u \in In$, are evenly spaced on Γ_u in the limit; it is proved by analyzing the behavior of calctarget as a discrete time dynamical system.

Lemma 4. *Consider a sequence of rounds $t_1 = T_{stab}, \ldots, t_n$. As $n \to \infty$, the locations of CNs in u, $u \in In$, are evenly spaced on Γ_u.*

Thus we conclude by summarizing the results in this section:

Theorem 1. *If there are no* fail *or* restart *actions for robots 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_u, $u \in U$, becomes fixed, and the size of the set is proportional to the quantized length q_u, within an a constant additive term $\frac{10(2m-1)}{q_{min}\rho_2}$.*
2. *All client nodes in a region $u \in U$ for which $q_u \neq 0$ are located on Γ_u and evenly spaced on Γ_u in the limit.*

6 Self-stabilization

In this section we show that the VSA-based motion coordination scheme is self-stabilizing. Specifically, we show that when the VSA and client components in the VSA layer start out in some arbitrary state (owing to failures and restarts), they eventually return to a reachable state. Thus, the visible behavior, or *traces*, of $VLayer[MC]$ running with some reachable state of $Vbcast \| RW \| VW$, eventually, becomes indistinguishable from a reachable trace of $VLayer[MC]$.

We first show that our motion coordination algorithm $VNodes[MC]$ is self-stabilizing to some set of legal states L_{MC}. Then, we show that these legal states correspond to reachable states of $VLayer[MC]$; hence, the traces of our motion coordination algorithm, where clients and VSAs start in an arbitrary state, eventually look like reachable traces of the correct motion coordination algorithm. Here MC is the motion coordination algorithm of Section 4.

6.1 Definitions and general results

We begin with some basic claims. Through out this section A, A_1, A_2, etc., are sets of actions and V is a set of variables. An (A, V)-sequence is a (possibly infinite) alternating sequence of actions in A and trajectories of V. Given (A, V)-sequences α, α' and $t \geq 0$, α' is a *t-suffix of α* if there exists a closed (A, V)-sequence α'' of duration t such that $\alpha = \alpha''\alpha'$. α' is a *state-matched t-suffix of α* if it is a t-suffix of α, and the first state of α' equals the last state of α''.

Given a set of (A_1, V)-sequences S_1, a set of (A_2, V)-sequences S_2, and $t \geq 0$, S_1 is said to *stabilizes in time* t to S_2 if each state-matched t-suffix α of each sequence in S_1 is in S_2. This *stabilizes to* relation is transitive as per the following:

Lemma 5. *Let S_i be a set of (A_i, V)-sequences, for $i \in \{1, 2, 3\}$. If S_1 stabilizes to S_2 in time t_1, and S_2 stabilizes to S_3 in time t_2, then S_1 stabilizes to S_3 in time $t_1 + t_2$.*

Let $\mathcal{A}$ be any TIOA with set of states $Q_{\mathcal{A}}$, and L be a nonempty subset of $Q_{\mathcal{A}}$. L is said to be a *legal set* for $\mathcal{A}$ if it is closed under the transitions and closed trajectories of $\mathcal{A}$. For any $L \subseteq Q_{\mathcal{A}}$, $Start(\mathcal{A}, L)$ is defined to be the TIOA that is identical to $\mathcal{A}$ except with starting states L. We define $U(\mathcal{A}) \triangleq Start(\mathcal{A}, Q_{\mathcal{A}})$ and $R(\mathcal{A}) \triangleq Start(\mathcal{A}, \mathsf{Reach}_{\mathcal{A}})$, where $\mathsf{Reach}_{\mathcal{A}}$ is the set of reachable states of $\mathcal{A}$.

Definition 2. *Let $\mathcal{B}$ and $\mathcal{A}$ be compatible TIOAs, and L be a legal set for the composed TIOA $\mathcal{A}\|\mathcal{B}$. $\mathcal{A}$* self-stabilizes in time t to L relative to $\mathcal{B}$ *if the set of executions of $U(\mathcal{A})\|\mathcal{B}$ stabilizes in time t to executions of $Start(\mathcal{A}\|\mathcal{B}, L)$.*

As per the theory of stabilizing emulations, assume we have a stabilizing VSA layer emulation such that each algorithm $alg \in VAlgs$ stabilizes in some t_{VStab} time to traces of $U(VLNodes[alg])\|R(RW\|VW\|Vbcast)$ that satisfy the additional property that for any $u \in U$, if there exists a client that has been in region u and alive for d_r time and no alive clients in the region failed or left in that time, then VSA V_u is not failed. In the context of this work, this means that if VSA layer algorithm MC is such that $VLNodes[MC]$ self-stabilizes in some time t to L_{MC} relative to $R(RW\|VW\|Vbcast)$, then we can conclude that physical node traces of the emulation algorithm on MC stabilize in time $t_{VStab} + t$ to client traces of executions of the VSA layer started in legal set L_{MC} and that satisfy the above failure-related properties.

6.2 Self-stabilization of our algorithm

We now describe two legal sets for $VLayer[MC]$, the second a subset of the first. The first is a set of states that results after the first $\mathsf{GPSupdate}$ at each client and the first time at each virtual node. It is easy to verify that this is a legal set.

Definition 3. *We define L^1_{MC} to be the set of states $x \in X_{VLayer[MC]}$ such that the following hold:*

1. $x\lceil X_{Vbcast\|RW\|VW} \in Reach_{Vbcast\|RW\|VW}$.
2. $\forall u \in U : \neg failed_u : clock_u \in \{RW.now, \perp\} \wedge (M_u \neq \emptyset \Rightarrow clock_u \mod \delta \in (0, e + 2d + 2\epsilon])$.
3. $\forall p \in P : \neg failed_p \Rightarrow \mathbf{v}_p \in \{RW.vel(p)/v_{max}, \perp\}$.
4. $\forall p \in P : \neg failed_p \wedge \mathbf{x}_p \neq \perp$:
   (a) $\mathbf{x}_p = RW.loc(p) \wedge clock_p = RW.now$.
   (b) $\mathbf{x}^*_p \in \{\mathbf{x}_p, \perp\} \vee ||\mathbf{x}^*_p - \mathbf{x}_p|| < v_{max}(\delta\lceil clock_p/\delta\rceil - clock_p - d_r)$.

(c) $Vbcast.reg(p) = region(\mathbf{x}_p) \vee clock \mod \delta \in (e+2d+2\epsilon, \delta - d_r + \epsilon_{sample})$.

Part 1 means that x restricted to the state of $Vbcast \| RW \| VW$ is a reachable state of $Vbcast \| RW \| VW$. Part 2 means that the nonfailed VSAs have *clock*s that are either equal to real-time or $\perp$, and have nonempty M only after the beginning of a round and up to $e + 2d + 2\epsilon$ time into a round. Part 3 requires that nonfailed clients have velocity vectors that are equal either to $\perp$ or equal to the client's velocity vector in RW, scaled down by v_{max}. Part 4 has three sub-parts and they assert that nonfailed clients with non-$\perp$ positions have (a) positions equal to their actual location and local *clock*s equal to the real-time, (b) targets equal to $\perp$ or the current location or a point reachable from the current location before a certain time (d_r), and (c) $Vbcast$ last region updates that match the current region or the time is within a certain time window in a round. The following stabilization result is also easy to verify.

Lemma 6. *$VLNodes[MC]$ is self-stabilizing to L^1_{MC} in time $t > \epsilon_{sample}$ relative to the automaton $R(Vbcast \| RW \| VW)$.*

The main legal set L_{MC} for our algorithm is described as the set of reachable states from a set of *reset* states.

Definition 4. *Define $Reset_{MC}$ to be the set of states $x \in X_{VLayer[MC]}$ such that the following properties hold:*

1. $x \in L^1_{MC}$.
2. $\forall p \in P : \neg failed_p \Rightarrow [tosnd_p^- = tosnd_p^+ = \lambda \wedge (\mathbf{x}_p = \perp \vee [\mathbf{x}_p^* \neq \perp \wedge \mathbf{v}_p = 0])]$.
3. $\forall u \in U : \neg failed_u \Rightarrow to\_send_u = \lambda$.
4. $\forall \langle m, u, t, P' \rangle \in vbcastq : P' = \emptyset$.
5. $RW.now \mod \delta = 0 \wedge \forall p \in P : \forall \langle l, t \rangle \in RW.updates(p) : t < RW.now$.

L_{MC} is the set of reachable states of $Start(VLayer[MC], Reset_{MC})$.

Part 2 states that each nonfailed client has empty queues in its $VBDelay$ and either has a position variable equal to $\perp$ or else has both a non-$\perp$ target and 0 velocity. Part 3 requires that each nonfailed VSA has an empty queue in its $VBDelay$. By Part 4 there are no pending messages in $Vbcast$, and Part 5 means that the time is the starting time for a round and that no GPSupdates have yet occurred at this time. It is easy to see that that L_{MC} is a legal set for *VLayer[MC]*. We show that starting from a state in L^1_{MC}, we reach a reset state which implies that eventually we arrive at a state in L_{MC}.

Lemma 7. *Executions of $VLayer[MC]$ started in states in L^1_{MC} stabilize in time $\delta + d + e$ to executions started in states in L_{MC}.*

Now we can combine our stabilization results to conclude that $VLNodes[MC]$ started in an arbitrary state and run with $R(Vbcast \| RW \| VW)$ stabilizes to L_{MC} in time $\delta + d + e + \epsilon_{sample}$. From transitivity of stabilization and 7, the next result follows.

Theorem 2. *$VLNodes[MC]$ is self-stabilizing to L_{MC} in time $\delta + d + e + \epsilon_{sample}$ relative to $R(Vbcast \| RW \| VW)$.*

6.3 Relationship between L_{MC} and reachable states

We just showed that $VLNodes[MC]$ is self-stabilizing to L_{MC} relative to the automaton $R(Vbcast \| RW \| VW)$. However, in order to conclude anything about the traces of $VLayer[MC]$ after stabilization, we need to show that traces of $VLayer[MC]$ starting in a state in L_{MC} are reachable traces of $VLayer[MC]$. We do this by first defining a simulation relation between states of $VLayer[MC]$ and then showing that for each state x in L_{MC} there is a reachable state y of $VLayer[MC]$ such that x is related to y under the simulation relation. This implies that the trace of any execution fragment starting with x is the trace of an execution fragment starting with y, which is a reachable trace of $VLayer[MC]$.

In order to show that each state in L_{MC} is related to some reachable state of $VLayer[MC]$, it is enough to show that each state in $Reset_{MC}$ is related to a reachable state of $VLayer[MC]$. The proof proceeds by providing a construction of an execution of $VLayer[MC]$ for each state in L_{MC}.

Lemma 8. *For each state $x \in Reset_{MC}$, there exists a reachable state y of $VLayer[MC]$ such that $x \mathcal{R}_{MC} y$.*

From these results it follows that the set of trace fragments of $VLayer[MC]$ starting from $Reset_{MC}$ is contained in the set of traces of $R(VLayer[MC])$. Bringing our results together we arrive at the main theorem:

Theorem 3. *The traces of $VLNodes[MC]$, starting in an arbitrary state and executed with automaton $R(Vbcast \| RW \| VW)$, stabilize in time $\delta + d + e + \epsilon_{sample}$ to reachable traces of $R(VLayer[MC])$.*

Thus, despite starting from an arbitrary configuration of the VSA and client components in the VSA layer, if there are no failures or restart of client nodes at or after some round t_0, then within a finite number of rounds after t_0, the clients are located on the curve and equally spaced in the limiting sense.

References

1. Dolev, S., Gilbert, S., Lahiani, L., Lynch, N.A., Nolte, T.A.: Virtual stationary automata for mobile networks. Technical Report MIT-LCS-TR-979 (2005)
2. Dolev, S., Gilbert, S., Lynch, N., Shvartsman, A., Welch, J.: Geoquorums: Implementing atomic memory in ad hoc networks. In: DISC 2003, Vol. 2848 of LNCS. (Oct 2003) 306–320
3. Dolev, S., Gilbert, S., Lynch, N.A., Shvartsman, A.A., Welch, J.: Geoquorums: Implementing atomic memory in mobile ad hoc networks. Distributed Computing (2005)
4. Chockler, G., Gilbert, S., Lynch, N.: Virtual infrastructure for collision-prone wireless networks. In: Proceedings of PODC. (2008) To appear.
5. Nolte, T., Lynch, N.A.: A virtual node-based tracking algorithm for mobile networks. In: ICDCS. (2007)
6. Dolev, S., Gilbert, S., Lahiani, L., Lynch, N., Nolte, T.: Virtual stationary automata for mobile networks. In: OPODIS. (2005)

7. Nolte, T., Lynch, N.A.: Self-stabilization and virtual node layer emulations. In: SSS. (2007) 394–408
8. Kaynar, D.K., Lynch, N., Segala, R., Vaandrager, F.: The Theory of Timed I/O Automata. Synthesis Lectures on Computer Science. Morgan Claypool (2005).
9. Jadbabaie, A., Lin, J., Morse, A.S.: Coordination of groups of mobile autonomous agents using nearest neighbor rules. IEEE Trans. on Automatic Control **48**(6) (2003) 988–1001
10. Ando, H., Oasa, Y., Suzuki, I., Yamashita, M.: Distributed memoryless point convergence algorithm for mobile robots with limited visibility. IEEE Trans. on Robotics and Automation **15**(5) (1999) 818–828
11. Lin, J., Morse, A., Anderson., B.: Multi-agent rendezvous problem. In: IEEE CDC 03. (2003)
12. Martinez, S., Cortes, J., Bullo, F.: On robust rendezvous for mobile autonomous agents. In: IFAC World Congress, (2005).
13. Gazi, V., Passino, K.M.: Stability analysis of swarms. IEEE Trans. on Automatic Control **48**(4) (2003) 692–697
14. Cortes, J., , Martinez, S., Karatas, T., Bullo, F.: Coverage control for mobile sensing networks. IEEE Trans. on Robotics & Automation **20**(2) (2004) 243–255
15. Suzuki, I., Yamashita, M.: Distributed autonomous mobile robots: Formation of geometric patterns. SIAM Journal of computing **28**(4) (1999) 1347–1363
16. Fax, J., Murray, R.: Information flow and cooperative control of vehicle formations. IEEE Trans. on Automatic Control **49** (2004) 1465–1476
17. Clavaski, S., Chaves, M., Day, R., Nag, P., Williams, A., Zhang, W.: Vehicle networks: achieving regular formation. In: ACC. (2003)
18. Blondel, V., Hendrickx, J., Olshevsky, A., Tsitsiklis, J.: Convergence in multiagent coordination consensus and flocking. In: IEEE CDC-ECC 05. (2005) 2996–3000
19. Olfati-Saber, R., Fax, J., Murray, R.: Consensus and cooperation in networked multi-agent systems. In: Proceedings of the IEEE **95**(1) (January 2007) 215–233
20. Prencipe, G.: Corda: Distributed coordination of a set of autonomous mobile robots. In: ERSADS. (May 2001 2001) 185–190
21. Flocchini, P., Prencipe, G., Santoro, N., Widmayer, P.: Pattern formation by autonomous robots without chirality. In: SIROCCO. (June 2001) 147–162
22. Efrima, A., Peleg, D.: Distributed models and algorithms for mobile robot systems. In: SOFSEM (1). LNCS 4362, Springer (January 2007) 70–87
23. Prencipe, G.: Achievable patterns by an even number of autonomous mobile robots. Technical Report TR-00-11 (2000)
24. Lynch, N., Mitra, S., Nolte, T.: Motion coordination using virtual nodes. In IEEE CDC05, (December 2005)
25. Brown, M.D.: Air traffic control using virtual stationary automata. Master's thesis, MIT (September 2007)