Safe Flocking in Spite of Actuator Faults

Taylor Johnson and Sayan Mitra

University of Illinois at Urbana-Champaign, Urbana, IL 61801, USA

Abstract. The safe flocking problem requires a collection of N mobile agents to (a) converge to and maintain an equi-spaced lattice formation, (b) arrive at a destination, and (c) always maintain a minimum safe separation. Safe flocking in Euclidean spaces is a well-studied and difficult coordination problem. Motivated by real-world deployment of multi-agent systems, this paper studies one-dimensional safe flocking, where agents are afflicted by *actuator faults*. An actuator fault is a new type of failure that causes an affected agent to be stuck moving with an arbitrary velocity. In this setting, first, a self-stabilizing solution for the problem is presented. This relies on a failure detector for actuator faults. Next, it is shown that certain actuator faults cannot be detected, while others may require $O(N)$ time for detection. Finally, a simple failure detector that achieves the latter bound is presented. Several simulation results are presented for illustrating the effects of failures on the progress towards flocking.

Keywords: failure detector, flocking, safety, stabilization, swarming

1 Introduction

Safe flocking is a distributed coordination problem that requires a collection of mobile agents situated in a Euclidean space to satisfy three properties, namely to: (a) form and maintain an equi-spaced lattice structure or a *flock*, (b) reach a specified destination or *goal* position, and (c) always maintain a minimum *safe* separation. The origins of this problem can be traced to biological studies aimed at understanding the rules that govern flocking in nature (see [13,11], for example). More recently, recognizing that such understanding could aid the design of autonomous robotic platoons or swarms, the problem as stated above and its variants have been studied in the robotics, control, and multi-agent systems literature (see [7,5,12,8,14] and references therein). Typically, the problem is studied for agents with synchronous communication, without failures, and with double-integrator dynamics—that is, the distributed algorithm sets the acceleration for each agent. To the best of our knowledge, even in this setting, safe-flocking is an open problem, as existing algorithms require unbounded accelerations for guaranteeing safety [12], which cannot be achieved in practice.

In this paper, we study one-dimensional safe-flocking within the realm of synchronous communication, but with a different set of dynamics and failure assumptions. First, we assume rectangular single-integrator dynamics. That is, at the beginning of each round, the algorithm decides a target point u_i for agent i based on messages received from i's neighbors, and agent i moves with bounded speed $\dot{x_i} \in [v_{min}, v_{max}]$ in the direction of u_i for the duration of that round. This simplifies the dynamics and achieving safety becomes relatively easy. Even in this setting however, it is nontrivial to develop and prove that an algorithm provides collision avoidance, as illustrated by an error—a forgotten case for the special dynamics of the rightmost (N^{th}) agent—that we found in the inductive proof of safety in [7]. To fix the error, the algorithm from [7] requires the modification presented later in this paper (Figure 3, Line 31). The model obtained with rectangular dynamics overapproximates any behavior that can be obtained with double integrator dynamics with bounded acceleration. Our algorithm combines the corrected algorithm from [7] with Chandy-Lamport's global snapshot algorithm [3]. The key idea is that each agent periodically computes its target based on messages received from its neighbors, then moves toward this target with some arbitrary but bounded velocity. The targets are computed such that the agents preserve safe separation and eventually form a *weak flock*, which remains invariant, and progress is ensured to a tighter *strong flock*. Once a strong flock is attained, this property can be detected through the use of a distributed snapshot algorithm [3]. Once this is detected, the detecting agent moves toward the destination, sacrificing the strong flock in favor of making progress toward the goal, but still preserving the weak flock.

Unlike the algorithms in [7,5,8,12] that provide convergence to a flock, we require the stronger *termination*. Our algorithm achieves termination through *quantization*: we assume that there exists a constant $\beta > 0$ such that an agent i moves in a particular round if and only if the computed target u_i is more than β away from the current position x_i. We believe that such quantized control is appropriate for realistic actuators, and useful for most power-constrained settings where it is undesirable for the agents to move forever in order to achieve convergence. Quantization affects the type of flock formation that we can achieve and also makes the proof of termination more interesting.

We allow agents to be affected by *actuator faults*. This physically corresponds to, for example, an agent's motors being stuck at an input voltage or a control surface becoming immobile. Actuator faults are permanent and cause the afflicted agents to move forever with a bounded and constant velocity. Actuator faults are a new class of failures that we believe are going to be important in designing and analyzing a wide range of distributed cyber-physical systems [9]. Unlike byzantine faults, behaviors resulting from actuator faults are constrained by physical laws. Also, unlike crash failures which typically thwart progress but not safety, actuator faults can also violate safety. A faulty agent has to be detected (and possibly avoided) by the non-faulty agents. In this paper, we assume that after an actuator fault, an agent continues to communicate and compute, but its actuators continue to move with the arbitrary but constant velocity.

Some attention has been given to failure detection in flocking such as [6], which works with a similar model of actuator faults. While [6] uses the therein developed *motion probes* in failure detection scenarios, no bounds are stated on detection time. Instead, convergence was ensured assuming that failure detection had occurred within some bounded time, while our work states an $O(N)$ detection time bound.

Our flocking algorithm determines *only* the direction in which an agent should move, based on the positions of adjacent agents. The speed with which an agent moves is chosen nondeterministically over a range, making the algorithm implementation independent with respect to the lower-level motion controller. Thus, the intuition behind failure detection is to observe that an agent has moved in the wrong direction. Under some assumptions about the system parameters, a simple lower-bound is established, indicating that no detection algorithm can detect failures in less than $O(N)$ rounds, where N is the number of agents. A failure detector is presented that utilizes this idea in detecting certain classes of failures in $O(N)$ rounds. Unfortunately, certain failures lead to a violation of safety in fewer rounds, so a failure detector which detects failures faster than $O(N)$ rounds is necessary to ensure safety. However, some failures are undetectable, such as an agent failing with zero velocity at the goal, and thus we establish that no such failure detector exists. But, under a restricted class of actuator faults, it is shown that the failure detector with $O(N)$ detection time can be combined with the flocking algorithm to guarantee the required safety and progress properties. This requires non-faulty agents to be able to avoid faulty ones. In one dimension (such as on highways), this is possible if there are multiple *lanes*.

In summary, the key contributions of the paper are the following:

(a) Formal introduction of the notion of actuator faults and stabilization in the face of such faults.
(b) A solution to the one-dimensional safe flocking problem in the face of actuator faults, quantization, and with bounded control. Our solution brings distributed computing ideas (self-stabilization and failure detection) to a distributed control problem.

2 System Model

This section presents a formal model of the distributed flocking algorithm modeled as a discrete transition system, as well as formal specifications of the system properties to be analyzed. For $K \in \mathbb{N}$, $[K] \triangleq \{1, \ldots, K\}$ and for a set S $S_\bot \triangleq S \cup \{\bot\}$. A *discrete transition system* $\mathcal{A}$ is a tuple $\langle X, Q, Q_0, A, \rightarrow\rangle$, where (i) X is a set of *variables* with associated types, (ii) Q is the set of *states*, which is the set of all possible valuations of the variables in X, (iii) $Q_0 \subseteq Q$ is the set of *start states*, (iv) A is a set of transition *labels*, and (v) $\rightarrow \subseteq Q \times A \times Q$ is a set of *discrete transitions*. An *execution fragment* of A is an (possibly infinite) alternating sequence of states and transition names, $\alpha = \mathbf{x}_0$, a_1, $\mathbf{x}_1$, ..., such that for each index k appearing in α, $(\mathbf{x}_k, a_{k+1}, \mathbf{x}_{k+1}) \in \rightarrow$. An *execution* is an execution fragment with $\mathbf{x}_0 \in Q_0$.

A state $\mathbf{x}$ is *reachable* if there exists a finite execution that ends in $\mathbf{x}$. A *stable* predicate $S \subseteq Q$ is a set of states closed under $\rightarrow$. If a stable predicate S contains Q_0, then it is called an *invariant* predicate and the reachable states of $\mathcal{A}$ are contained in S. A *safety* property specified by a predicate $S \subseteq Q$ is satisfied by $\mathcal{A}$ if all of its reachable states are contained in S. Self-stabilization is a property of non-masking fault tolerance which guarantees that once new failures cease to occur, the system eventually returns to a legal state [4]. In this paper, we model actuator faults by transitions with the special label fail. Given $G \subseteq Q$, $\mathcal{A}$ *self-stabilizes* to G if (a) G is a stable predicate for $\mathcal{A}$ along execution fragments without fail-transitions, and (b) from every reachable state of $\mathcal{A}$ (including states reached via fail transitions), every *fail*-free execution fragment eventually reaches G.

2.1 Model of Safe Flocking System

The distributed system consists of a set of N mobile *agents* physically positioned on N_L infinite, parallel *lanes*. The system can be thought of as a collection of cars in the lanes on a highway. Refer to Figure 1 for clarity, and Figure 2 shows the system making progress (reaching the origin as a flock), without indicating lanes, but note that agents 1 through 5 move to lane 2 around round 375 to avoid the failed agent 6. We assume synchrony and the communication graph is complete, regardless of lanes[1]. That is, agents have synchronized clocks, message delays are bounded, and computations are instantaneous. At each round, each agent exchanges messages bearing state information with everyone, and note that this means agents in different lanes communicate. Agents then update their software state and (nondeterministically) choose their velocities, which they operate with until the beginning of the next round. Under these assumptions, it is convenient to model the system as a collection of discrete transition systems that interact through shared variables. Let $ID \triangleq [N]$ be the set of unique agent identifiers and $LD \triangleq [N_L]$ be the set of lane identifiers. The following positive constants are used throughout the paper: (a) r_s: minimum required inter-agent gap or *safety distance* in the absence of failures, (b) r_r: reduced safety distance in the presence of failures, (c) r_f: desired maximum inter-agent gap which defines a flock, (d) δ: flocking tolerance parameter—that is, the maximum deviation from r_f agents may be spaced and constitute a *flock*, (e) β: quantization parameter, used to prevent agents from moving if the algorithm decides too small a movement so that eventually the algorithm terminates, and (f) v_{min}, v_{max}: minimum and maximum velocities.

State Variables. The discrete transition system corresponding to $\mathsf{Agent_i}$ has the following *private* variables with the *type followed by initial value in parentheses*: (a) gsf (Boolean; $false$): indicates whether the stable predicate detected by the global snapshot is satisfied or not, (b) sr (Boolean; $false$): indicates whether the

[1] This communication assumption is relaxed to nearby neighbors being able to communicate synchronously in [9].

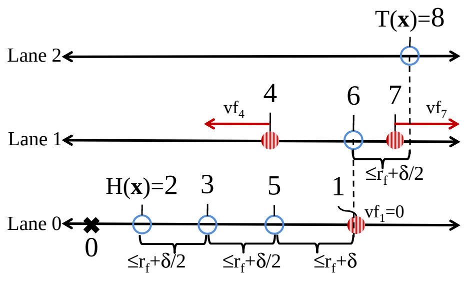

Fig. 1. System at state $\mathbf{x}$ for $N = 8$, $\bar{F}(\mathbf{x}) = \{2, 3, 5, 6, 8\}$, $F(\mathbf{x}) = \{1, 4, 7\}$. Failed actuator velocities are labeled vf_i and eventually 4 and 7 will diverge. Non-faulty agents have avoided failed agents by changing lanes. Note that $L(\mathbf{x}, 6) = 5$. Also, if $4 \in Suspected_6$, then $L_S(\mathbf{x}, 6) = L(6) = 5$, else $L_S(\mathbf{x}, 6) = 4$. Assuming $S(\mathbf{x}) = F(\mathbf{x})$, $Flock_W(\mathbf{x})$, but $\neg Flock_S(\mathbf{x})$, since $|\mathbf{x}.x_6 - \mathbf{x}.x_5 - r_f| \leq \delta$ (and not $\delta/2$).

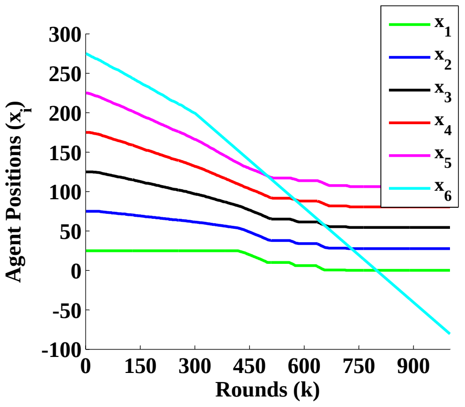

Fig. 2. System progressing: eventually the agents have formed a flock and the failed agent 6 with nonzero velocity has diverged.

global snapshot algorithm has been initiated, (c) *failed* (Boolean; *false*): indicates whether or not agent i has failed, (d) vf ($\mathbb{R}$; $\perp$): velocity with which agent i has failed, and (e) L and R ($ID_\perp$): identifiers of the nearest left and right agents to agent i. The following *shared* variables are controlled by agent i, but can also be read by others (*type followed by initial value in parentheses*): (a) x and xo ($\mathbb{R}$): current position and position from the previous round of agent i, (b) u and uo ($\mathbb{R}$; x and xo): target position and target position from the previous round of agent i, (c) $lane$ (LD; 1): the lane currently occupied by agent i, (d) $Suspected$ ($ID_\perp$; $\emptyset$): set of neighbors that agent i believes to have failed. These variables are *shared* in the following sense: At the beginning of each round k, their values are broadcast by Agent$_i$ and are used by other agents to update their states in that round. The discrete transition system modeling the complete ensemble of agents is called System. We refer to states of System with bold letters $\mathbf{x}$, $\mathbf{x}'$, etc., and Agent$_i$'s individual state components by $\mathbf{x}.x_i$, $\mathbf{x}.u_i$, etc.

Actuator Faults and Failure Detection. The failure of agent i's actuators is modeled by the occurrence of a transition labeled by fail$_i$. This transition is always enabled unless i has already failed, and as a result of its occurrence, the variable $failed_i$ is set to $true$. An actuator fault causes the affected agent to move forever with a constant but arbitrary *failure velocity*. At state $\mathbf{x}$, $F(\mathbf{x})$ and $\bar{F}(\mathbf{x})$ denote the sets of faulty and non-faulty agent identifiers, respectively.

Agents do not have any direct information regarding the failure of other agents' actuators (i.e., agent i cannot read $failed_j$). Agents rely on timely failure detection to avoid violating safety or drifting away from the goal by following a faulty agent. Failure detection at agent i is abstractly captured by the $Suspected_i$ variable and a transition labeled by suspect$_i$. The suspect$_i(j)$ transition models a detection of failure of some agent j by agent i. Failures are irreversible in our model, and thus so are failure detector suspicions. For agent i, at any given state $Suspected_i \subseteq ID$ is the set of agent identifiers that agent i's failure detector suspects as faulty. Agent$_j$ is said to be *suspected* if some agent i suspects it, otherwise it is *unsuspected*. Which particular agent suspects a faulty agent j is somewhat irrelevant. We assume the failure detectors of all agents share information through some background gossip, and when one agent suspects agent i, all other agents also suspect i in the same round[2]. Denote the sets of suspected and unsuspected agents by $S(\mathbf{x})$ and $\bar{S}(\mathbf{x})$, respectively.

The *detection time* is the minimum number of rounds within which every failure is always suspected. In most parts of Section 3 we will assume that there exists a finite detection time k_d for any failure. In Section 3.3, we will discuss specific conditions under which k_d is in fact finite and then give upper and lower bounds for it. The failure detection strategy used by our flocking algorithm is encoded as the precondition of the suspect transition. Note that the precondition assumes that i has access to some of j's shared variables, namely x_j, xo_j, u_j and uo_j. When the precondition of suspect(j) is satisfied at Figure 3, j is added to $Suspected_i$. This precondition checks that either j moved when it should not have, or that j moved in the wrong direction, away from its computed target. The rationale behind this condition will become clear as we discuss the flocking algorithm.

```
fail$_i(v)$, $|v| \leq v_{max}$
pre $\neg$ $failed$
eff $failed := true; vf := v$

suspect$_i(j)$, $j \in ID$
pre $j \notin Suspected \wedge ($if $|xo_j - uo_j| \geq \beta$
    then sgn $(x_j - xo_j) \neq$ sgn $(uo_j - xo_j)$
    else $|x_j - uo_j| \neq 0)$
eff $Suspected := Suspected \cup \{j\}$

snapStart$_i$
pre $L = \perp \wedge \neg sr$
eff $sr := true$ // global snapshot invoked

snapEnd$_i(GS)$, $GS \in \{false, true\}$
eff $gsf := GS$; // global snapshot returns
    $sr := false$

update$_i$
eff $uo := u; xo := x$
  for each $j \in ID$, $Suspected := Suspected \cup Suspected_j$
$Mitigate$:
  if $\neg$ $failed \wedge (\exists\, s \in Suspected : lane_s = lane)$
    $\wedge (\exists\, \mathcal{L} \in LD : \forall\, j \in ID, (lane_j = \mathcal{L} \Rightarrow$
    $x_j \notin [x - r_s - 2v_{max}, x + r_s + 2v_{max}]))$
    then $lane := \mathcal{L}$ fi
$Target$:
  if $L = \perp \wedge gsf$ then $u := x - \min\{x, \delta/2\}$;
    $gsf := false$
  elseif $L = \perp$ then $u := x$
  elseif $R = \perp$ then $u := (x_L + x + r_f)/2$
  else $u := (x_L + x_R)/2$ fi
$Quant$: if $|u - x| < \beta$ then $u := x$ fi
$Move$: if $failed$ then $x := x + vf$
  else $x := x +$ sgn $(x - u)$ choose $[v_{min}, v_{max}]$ fi
```

Fig. 3. Agent$_i$'s transitions: failure detection, global snapshots, and target updates.

Neighbors. At state $\mathbf{x}$, let $L(\mathbf{x}, i)$ (and symmetrically $R(\mathbf{x}, i)$) be the nearest non-failed agent left (resp. right) of Agent$_i$, with ties broken arbitrarily. If no such agent exists, then $L(\mathbf{x}, i)$ and $R(\mathbf{x}, i)$ are defined as $\perp$. Let $L_S(\mathbf{x}, i)$ (and symmetrically $R_S(\mathbf{x}, i)$) be the nearest unsuspected agent left (resp. right) of Agent$_i$ at state $\mathbf{x}$, or $\perp$ if no such agents exist. An unsuspected Agent$_i$ with both unsuspected left and right neighbors is a *middle agent*. An unsuspected Agent$_i$ without an unsuspected left neighbor is the *head agent*, and is denoted by the singleton $H(\mathbf{x})$. If Agent$_i$ is unsuspected, is not the head, and does not have an unsuspected right neighbor, it is the *tail agent* and is denoted by the singleton $T(\mathbf{x})$.

[2] This assumption is relaxed to adjacent agents in [9].

Flocking Algorithm. The distributed flocking algorithm executed at $\mathsf{Agent_i}$ uses two separate processes (threads): (a) a process for taking distributed global snapshots, and (b) a process for updating the target position for $\mathsf{Agent_i}$.

The snapStart and snapEnd transitions model the periodic initialization and termination of a distributed global snapshot protocol—such as Chandy and Lamport's snapshot algorithm [3]—by the head agent. This global snapshot is used for detecting a stable global predicate, which in turn influences the target computation for the head agent. Although we have not modeled this explicitly, we assume that the $\mathsf{snapStart}_i$ transition is performed periodically by the head agent when the precondition is enabled. If the global predicate holds, then $\mathsf{snapEnd}(true)$ occurs, otherwise $\mathsf{snapEnd}(false)$ occurs. Chandy-Lamport's algorithm can be applied since (a) we are detecting a stable predicate, (b) the communications graph is complete, and (c) the stable predicate being detected is reachable. Thus, we assume that in any infinite execution, a $\mathsf{snapEnd}_i$ transition occurs within $O(N)$ rounds from the occurrence of the corresponding $\mathsf{snapStart}_i$ transition.

The update transition models the evolution of all (faulty and non-faulty) agents over a synchronous round. It is composed of four subroutines: $Mitigate$, $Target$, $Quant$, and $Move$, which are executed in this sequence for updating the state of System. The entire update is instantaneous and atomic; the subroutines are used for clarity of presentation. To be clear, for $\mathbf{x} \overset{\mathsf{update}}{\rightarrow} \mathbf{x}'$, $\mathbf{x}'$ is obtained by applying each of these subroutines. We refer to the intermediate states after $Mitigate$, $Target$, $Quant$, and $Move$ as $\mathbf{x}_M$, $\mathbf{x}_T$, $\mathbf{x}_Q$, and $\mathbf{x}_V$, respectively. That is, $\mathbf{x}_M \triangleq Mitigate(\mathbf{x})$, $\mathbf{x}_T \triangleq Target(\mathbf{x}_M)$, etc., and note $\mathbf{x}' = \mathbf{x}_V = Move(\mathbf{x}_Q)$.

$Mitigate$ is executed by non-faulty agents and may cause them to change lanes, thus restoring safety and progress properties that may be reduced or violated by failures. $Target$ determines a new target to move toward. There are three different rules for target computations based on an agent's belief of whether it is a head, middle, or tail agent. For a state $\mathbf{x}$, each middle agent i attempts to maintain the average of the positions of its nearest unsuspected left and right neighbors (Figure 3, Line 32). Assuming that the goal is to the left of the tail agent, the tail agent attempts to maintain r_f distance from its nearest unsuspected left neighbor (Figure 3, Line 31). The head agent periodically invokes a global snapshot and attempts to detect a certain stable global predicate $Flock_S$ (defined below). If this predicate is detected, then the head agent moves towards the goal (Figure 3, Line 29), otherwise it does not change its target u from its current position x. As mentioned before, targets are still computed for faulty agents, but their actuators ignore these new values. $Quant$ is the quantization step which prevents targets u_i computed in the $Target$ subroutine from being applied to real positions x_i, if the difference between the two is smaller than the *quantization parameter* β. It is worth emphasizing that quantization is a key requirement for any realistic algorithm that actuates the agents to move with bounded velocities. Without quantization, if the computed target is very close to the current position of the agent, then the agent may have to move with arbitrarily small velocity over that round. Finally, $Move$ moves agent positions

x_i toward the quantized targets. Note that Move abstractly captures the physical evolution of the system over a round; that is, it is the time-abstract transition corresponding to physical evolution over an interval of time.

2.2 Key Predicates

We now define a set of predicates on the state space of System that capture the key properties of safe flocking. These will be used for proving that the algorithm described above solves safe flocking in the presence of actuator faults. We start with safety. A state $\mathbf{x}$ of System satisfies Safety if the distance between every pair of agents on the same lane is at least the safety distance r_s. Formally, $\mathit{Safety}(\mathbf{x}) \triangleq \forall i, j \in ID, i \neq j, \mathbf{x}.lane_i = \mathbf{x}.lane_j \implies |\mathbf{x}.x_i - \mathbf{x}.x_j| \geq r_s$. When failures occur, a reduced inter-agent gap of r_r will be guaranteed. We call this weaker property *reduced safety*: $\mathit{Safety}_R(\mathbf{x}) \triangleq \forall i \in \bar{F}(\mathbf{x}), \forall j \in ID, i \neq j, \mathbf{x}.lane_i = \mathbf{x}.lane_j \implies |\mathbf{x}.x_i - \mathbf{x}.x_j| \geq r_r$.

An ϵ-flock is where each non-faulty agent with an unsuspected left neighbor (not necessarily in the same lane) is within $r_f \pm \epsilon$ from that neighbor. Formally, $\mathit{Flock}(\mathbf{x}, \epsilon) \triangleq \forall i \in \bar{S}(\mathbf{x}), L_S(\mathbf{x}, i) \neq \bot, \left|\mathbf{x}.x_i - \mathbf{x}.x_{L_S(\mathbf{x},i)} - r_f\right| \leq \epsilon$. In this paper, we will use the Flock predicate with two specific values of ϵ, namely δ (the flocking tolerance parameter) and $\frac{\delta}{2}$. The *weak flock* and the *strong flock* predicates are defined as $\mathit{Flock}_W(\mathbf{x}) \triangleq \mathit{Flock}(\mathbf{x}, \delta)$, and $\mathit{Flock}_S(\mathbf{x}) \triangleq \mathit{Flock}(\mathbf{x}, \frac{\delta}{2})$, respectively.

Related to quantization, we have the *no big moves (NBM)* predicate, where none of the agents (except possibly the head agent) have any valid moves, because their computed targets are less than β (quantization constant) away from their current positions. $\mathit{NBM}(\mathbf{x}) \triangleq \forall i \in \bar{F}(\mathbf{x}), L_S(\mathbf{x}, i) \neq \bot, |\mathbf{x}_T.u_i - \mathbf{x}.x_i| \leq \beta$, where $\mathbf{x}_T$ is the state following the application of Target subroutine to $\mathbf{x}$. The Goal predicate is satisfied at states where the head agent is within β distance of the goal (assumed to be the origin without loss of generality), that is, $\mathit{Goal}(\mathbf{x}) \triangleq \mathbf{x}.x_{H(\mathbf{x})} \in [0, \beta)$. Finally, a state satisfies the $\mathit{Terminal}$ predicate if it satisfies both Goal and NBM.

3 Analysis

The main result of the paper (Theorem 1) is that the algorithm in Figure 3 achieves safe flocking in spite of failures provided: (a) there exists a failure detector that detects actuator faults *sufficiently fast*, and (b) each non-faulty agent has *enough room* to jump to some lane to safely avoid faulty agents and eventually make progress. For the first part of our analysis, we will simply assume that any failure is detected within k_d rounds. In Section 3.3, we shall examine conditions under which k_d is finite and state its lower and upper bounds. Assumption (b) is trivially satisfied if the number of lanes is greater than the total number of failures; but it is also satisfied with fewer lanes, provided the failures are sufficiently apart in space. There are two space requirements for Assumption (b): the first ensures safety and the second ensure progress by preventing "walls" of faulty agents from existing forever and ensuring that infinitely often all non-faulty agents may make progress.

Theorem 1. *Suppose there exists a failure detector which suspects any actuator fault within k_d rounds. Suppose further that $v_{max} \leq (r_s - r_r)/(2k_d)$. Let $\alpha = \mathbf{x}_0, \ldots, \mathbf{x}_p, \mathbf{x}_{p+1}$ be an execution where x_p is the state after the last* fail *transition. Let $\alpha_{ff} = \mathbf{x}_{p+1}, \ldots$, be the fail-free suffix of α. Let f be the number of actuator faults. Suppose either (a) $N_L > f$,* or *(b) $N_L \leq f$ and along α_{ff}, $\forall \mathbf{x} \in \alpha_{ff}$, $\exists \mathcal{L} \in LD$ such that $\forall i \in \bar{F}(\mathbf{x})$, $\forall j \in F(\mathbf{x})$, $\mathbf{x}.lane_j \neq \mathcal{L}$ and $|\mathbf{x}.x_i - \mathbf{x}.x_j| > r_s + 2v_{max}k_d$, and also that infinitely often, $\forall m, n \in F(\mathbf{x})$, $m \neq n$, $|\mathbf{x}.x_m - \mathbf{x}.x_n| > r_s + 2v_{max}$. Then, (a) Every state in α satisfies the reduced safety property, $Safety_R$, and (b) Eventually $Terminal$ and $Flock_S$ are satisfied.*

In what follows, we state and informally discuss a sequence of lemmas that culminate in Theorem 1. Under the assumptions and analysis of this section, the following relationships are satisfied: $NBM \subset Flock_S \subset Flock_W \subset Safety \subset Safety_R$. Detailed proofs of the lemmas appear in the technical report [10]. We begin with some assumptions.

Assumptions. Except where noted in Section 3.3, the remainder of the paper utilizes the assumptions of Theorem 1. Additionally, these assumptions are required throughout the paper: (a) $N_L \geq 2$: there are at least 2 lanes, (b) $r_r < r_s < r_f$: the reduced safety gap r_r required under failures is strictly less than the safety gap r_s in the absence of failures, which in turn is strictly less than the flocking distance, (c) $0 < v_{min} \leq v_{max} \leq \beta \leq \delta/(4N)$, and (d) the communication graph of the non-faulty agents is always fully connected, so the graph of non-faulty agents cannot partition. Assumption (c) bounds the minimum and maximum velocities, although they may be equal. It then upper bounds the maximum velocity to be less than or equal to the quantization parameter β. This is necessary to prevent a violation of safety due to overshooting computed targets. Finally, β is upper bounded such that $NBM \subseteq Flock_S$. Intuitively, the bound on β is to ensure that errors from flocking due to quantization do not accumulate along the flock from the head to the tail. This is used to show that eventually $Flock_S$ is satisfied by showing eventually NBM is reached.

3.1 Safety

First, we establish that System satisfies the safety part of the safe flocking problem. The following lemma states that in each round, each agent moves by at most v_{max}, and follows immediately from the specification of System.

Lemma 1. *For any two states $\mathbf{x}, \mathbf{x}'$ of* System, *if $\mathbf{x} \xrightarrow{a} \mathbf{x}'$ for some transition a, then for each agent $i \in ID$, $|\mathbf{x}'.x_i - \mathbf{x}.x_i| \leq v_{max}$.*

The next lemma establishes that, upon changes in which other agents an agent i uses to compute its target position, safety is not violated.

Lemma 2. *For any execution α, for states $\mathbf{x}, \mathbf{x}' \in \alpha$ such that $\mathbf{x} \xrightarrow{a} \mathbf{x}'$ for any $a \in A$, $\forall i, j \in ID$, if $L_S(\mathbf{x}, i) \neq j$ and $R_S(\mathbf{x}, j) \neq i$ and $L_S(\mathbf{x}', i) = j$ and $R_S(\mathbf{x}', j) = i$ and $\mathbf{x}.x_{R_S(\mathbf{x},j)} - \mathbf{x}.x_{L_S(\mathbf{x},i)} \geq c$, then $\mathbf{x}'.x_{R_S(\mathbf{x}',j)} - \mathbf{x}'.x_{L_S(\mathbf{x}',i)} \geq c$, for any $c > 0$.*

Invariant 1 shows the spacing between any two non-faulty agents in any lane is always at least r_r, and the spacing between any non-faulty agent and any other agent in the same lane is at least r_r. There is no result on the spacing between any two faulty agents—they may collide. The proof is by induction.

Invariant 1. *For any reachable state $\mathbf{x}$, $Safety_R(\mathbf{x})$.*

3.2 Progress

The progress analysis works with fail-free executions, that is, there are no further fail_i transitions. Note that this does not mean $F(\mathbf{x}) = \emptyset$, only that along such executions $|F(\mathbf{x})|$ does not change. This is a standard assumption used to show convergence from an arbitrary state back to a stable set [1], albeit we note that we are dealing with permanent faults instead of transient ones. In this case, the stable set eventually reached are states where *Terminal* is satisfied. However, note that the first state in such an execution is not entirely arbitrary, as Section 3.1 established that such states satisfy at least *Safety*$_R$, and all the following analysis relies on this assumption.

First observe that, like safety, progress may be violated by failures. Any failed agent with nonzero velocity diverges by the definition of velocities in Figure 3, Line 34. This observation also highlights why *Flock* is quantified over agents with identifiers in the set of suspected agents $\bar{S}(\mathbf{x})$ and not the set of failed agents $\bar{F}(\mathbf{x})$ or all agents ID—if it were quantified over ID, at no future point could $Flock(\mathbf{x})$ be attained if a failed agent has diverged. Zero velocity failures may also cause progress to be violated, where a "wall" of non-moving failed agents may be created, but such situations are excluded by the second part of Assumption (b) in Theorem 1.

Progress along Fail-Free Executions. In the remainder of this section, we show that once new actuator faults cease occurring, System eventually reaches a state satisfying *Terminal*. This is a convergence proof and we will use a Lyapunov-like function to prove this property. The remainder of this section applies to any infinite fail-free execution fragment, so fix such a fragment α_{ff}.

These descriptions of error dynamics are used in the analysis:

$$e(\mathbf{x}, i) \triangleq \begin{cases} |\mathbf{x}.x_i - \mathbf{x}.x_{\mathbf{x}.L_i} - r_f| & \text{if } i \text{ is a middle or a tail agent,} \\ 0 & \text{otherwise,} \end{cases}$$

$$eu(\mathbf{x}, i) \triangleq \begin{cases} |\mathbf{x}.u_i - \mathbf{x}.u_{\mathbf{x}.L_i} - r_f| & \text{if } i \text{ is a middle or a tail agent,} \\ 0 & \text{otherwise.} \end{cases}$$

Here $e(\mathbf{x}, i)$ gives the error with respect to r_f of Agent_i and its non-suspected left neighbor and $eu(\mathbf{x}, i)$, with respect to target positions $\mathbf{x}.u_i$ rather than physical positions $\mathbf{x}.x_i$.

Now, we make the simple observation from Line 35 of Figure 3 that if a non-faulty agent i moves in some round, then it moves by at least a positive amount v_{min}. Observe that an agent may not move in a round if the conditional in Figure 3, Line 33 is satisfied, but this does not imply $v_{min} = 0$. Then, Lemma 3 states that from any reachable state $\mathbf{x}$ which does not satisfy NBM, the maximum error over all non-faulty agents in non-increasing. This is shown by first noting that only the update transition can cause any change of $e(\mathbf{x}, i)$ or $eu(\mathbf{x}, i)$, and then analyzing the change in value of $eu(\mathbf{x}, i)$ for each of the computations of u_i in the $Target$ subroutine of the update transition. Then it is shown that applying the $Quant$ subroutine cannot cause any $eu(\mathbf{x}, i)$ to increase, and finally computing x_i in the $Move$ subroutine does not increase any $e(\mathbf{x}, i)$.

Lemma 3. *For reachable states $\mathbf{x}, \mathbf{x}'$, if $\mathbf{x} \xrightarrow{a} \mathbf{x}'$ and $\mathbf{x} \notin NBM$, for some $a \in A$, then $\max\limits_{i \in \bar{F}(\mathbf{x})} e(\mathbf{x}', i) \leq \max\limits_{i \in \bar{F}(\mathbf{x})} e(\mathbf{x}, i)$.*

Next, Lemma 4 shows sets of states satisfying NBM are invariant, a state satisfying NBM is reached, and gives a bound on the number of rounds required to reach such a state. Define the candidate Lyapunov function as $V(\mathbf{x}) \triangleq \sum_{i \in \bar{F}(\mathbf{x})} e(\mathbf{x}, i)$. Define the maximum value the candidate Lyapunov function obtained over any state $\mathbf{x} \in \alpha_{ff}$ satisfying NBM as $\gamma \triangleq \sup\limits_{\mathbf{x} \in NBM} V(\mathbf{x})$.

Lemma 4. *Let $\mathbf{x}_k$ be the first state of α_{ff}, and let the head agent's position be fixed. If $V(\mathbf{x}_k) > \gamma$, then the* update *transition decreases $V(\mathbf{x}_k)$ by at least a positive constant ψ. Furthermore, there exists a finite round c such that $V(\mathbf{x}_c) \leq \gamma$, where $\mathbf{x}_c \in NBM(\mathbf{x})$ and $k < c \leq \lceil (V(\mathbf{x}_k) - \gamma)/\psi \rceil$, where $\psi = v_{min}$.*

Lemma 4 stated a bound on the time it takes for System to reach the set of states satisfying NBM. However, to satisfy $Flock_S(\mathbf{x})$, all $\mathbf{x} \in NBM$ must be inside the set of states that satisfy $Flock_S$, and the following lemma states this. From any state $\mathbf{x}$ that does not satisfy $Flock_S(\mathbf{x})$, there exists an agent that computes a control that will satisfy the quantization constraint and hence make a move towards NBM. This follows from the assumption that $\beta \leq \delta/(4N)$.

Lemma 5. *If $Flock_S(\mathbf{x})$, then $V(\mathbf{x}) \leq \sum_{i \in \bar{F}(\mathbf{x})} e(\mathbf{x}, i) = (\delta \left|\bar{F}(\mathbf{x})\right|)/4$.*

Now we observe that $Flock_W$ is a stable predicate, that is, that once a weak flock is formed, it remains invariant. This result follows from analyzing the $Target$ subroutine which computes the new targets for the agents in each round. Note that the head agent moves by a fixed distance $\frac{\delta}{2}$, only when $Flock_S$ holds, which guarantees that $Flock_W$ is maintained even though $Flock_S$ may be violated. This establishes that for any reachable state $\mathbf{x}'$, if $V(\mathbf{x}') > V(\mathbf{x})$, then $V(\mathbf{x}') < (\delta \left|\bar{F}(\mathbf{x})\right|)/2$.

Lemma 6. *$Flock_W$ is a stable predicate.*

The following corollary follows from Lemma 4, as $Flock_S(\mathbf{x})$ is violated after becoming satisfied only if the head agent moves, in which case $\mathbf{x}'.x_{H(\mathbf{x}')} < \mathbf{x}.x_{H(\mathbf{x})}$, which causes $V(\mathbf{x}') \geq V(\mathbf{x})$.

Corollary 1. *For $\mathbf{x} \in \alpha_{ff}$ such that, if $Flocks_S(\mathbf{x})$, $\mathbf{x} \xrightarrow{a} \mathbf{x}'$ $\forall a \in A$, and $\mathbf{x}.x_{H(\mathbf{x})} = \mathbf{x}'.x_{H(\mathbf{x}')}$, then $Flocks_S(\mathbf{x}')$.*

The following lemma—with Assumption (b) of Theorem 1 that gives eventually a state is reached such that non-faulty agents may pass faulty agents—is sufficient to prove that $Terminal$ is eventually satisfied in spite of failures. After this number of rounds, no agent $j \in \bar{F}(\mathbf{x})$ believes any $i \in F(\mathbf{x})$ is its left or right neighbor, and thereby any failed agents diverge safely along their individual lanes if $|\mathbf{x}.v_i| > 0$ by the observation that failed agents with nonzero velocity diverge. Particularly, after some agent j has been suspected by all non-faulty agents, the $Mitigate$ subroutine of the update transition shows that the non-faulty agents will move to a different lane at the next round. This shows that mitigation takes at most one additional round after detection, since we have assumed in Theorem 1 that there is always free space on some lane. This implies that so long as a failed agent is detected prior to safety being violated, only one additional round is required to mitigate, so the time of mitigation is a constant factor added to the time to suspect, resulting in the constant c being linear in the number of agents.

Lemma 7. *For any fail-free execution fragment α_{ff}, if $\mathbf{x}.failed_i$ at some state $\mathbf{x} \in \alpha_{ff}$, then for a state $\mathbf{x}' \in \alpha_{ff}$ at least c rounds from $\mathbf{x}$, $\forall j \in ID.\mathbf{x}'.L_j \neq i \wedge \mathbf{x}'.R_j \neq i$.*

The next theorem shows that System eventually reaches the goal as a strong flock, that is, there is a finite round t such that $Terminal(\mathbf{x}_t)$ and $Flocks_S(\mathbf{x}_t)$ and shows that System is self-stabilizing when combined with a failure detector.

Theorem 2. *Let α_{ff} be written $\mathbf{x}_0, \mathbf{x}_1, \ldots$. Consider the infinite sequence of pairs $\langle \mathbf{x}_0.x_{H(\mathbf{x}_0)}, V(\mathbf{x}_0)\rangle$, $\langle \mathbf{x}_1.x_{H(\mathbf{x}_1)}, V(\mathbf{x}_1)\rangle$, $\ldots$, $\langle \mathbf{x}_t.x_{H(\mathbf{x}_t)}, V(\mathbf{x}_t)\rangle$, $\ldots$. Then, there exists t at most $\left\lceil \frac{(V(\mathbf{x}_0) - |\bar{F}(\mathbf{x})|\delta/4)}{v_{min}} \right\rceil + \left\lceil \frac{|\bar{F}(\mathbf{x})|\delta/4}{v_{min}} \right\rceil \max\{1, \frac{\mathbf{x}_0.x_{H(\mathbf{x}_0)}}{v_{min}} O(N)\}$ rounds from $\mathbf{x}_0$ in α_{ff}, such that: (a) $\mathbf{x}_t.x_{H(\mathbf{x}_t)} = \mathbf{x}_{t+1}.x_{H(\mathbf{x}_{t+1})}$, (b) $V(\mathbf{x}_t) = V(\mathbf{x}_{t+1})$, (c) $\mathbf{x}_t.x_{H(\mathbf{x}_t)} \in [0, \beta]$, (d) $V(\mathbf{x}_t) \leq |\bar{F}(\mathbf{x})| \frac{\delta}{4}$, (e) $Terminal(\mathbf{x}_t)$, and (f) $Flocks_S(\mathbf{x}_t)$.*

3.3 Failure Detection

In the earlier analysis we assumed that it is possible to detect all actuator faults within finite number of rounds k_d. Unfortunately this is not true, as there exist failures which cannot be detected at all. A trivial example of such an undetectable failures is the failure of a node with 0 velocity at a terminal state, that is, a state at which all the agents are at the goal in a flock and therefore are static. While such failures were undetectable in any number of rounds, these failures do not violate $Safety$ or $Terminal$. It turns out that only failures which cause a violation of safety or progress may be detected.

Lower-Bound on Detection Time. While the occurrence of $\mathsf{fail}_i(v)$ may never be detected in some cases as just illustrated, we show a lower-bound on the detection time for all $\mathsf{fail}_i(v)$ transitions that can be detected. The following lower-bound applies for executions beginning from states that do not *a priori* satisfy

$Terminal$. It says that a failed agent mimicked the actions of its correct non-faulty behavior in such a way that despite the failure, System still progressed to NBM as was intended. From an arbitrary state, it takes $O(N)$ rounds to converge to a state satisfying NBM by Lemma 4.

Lemma 8. *The detection time lower-bound for any detectable actuator fault is $O(N)$.*

Next we show that the the failure detection mechanism incorporated in Figure 3 does not produce any false positives.

Lemma 9. *In any reachable state $\mathbf{x}$, $\forall j \in \mathbf{x}.Suspected_i \Rightarrow \mathbf{x}.failed_j$.*

The next lemma shows a partial *completeness* property [2] of the failure detection mechanism incorporated in Figure 3.

Lemma 10. *Suppose that $\mathbf{x}$ is a state in the fail-free execution fragment α_{ff} such that $\exists$ $j \in F(\mathbf{x})$, $\exists\, i \in ID$, and j is not suspected by i. Suppose that either (a) $|\mathbf{x}.xo_j - \mathbf{x}.uo_j|$ $\leq \beta$ and $|\mathbf{x}.x_j - \mathbf{x}.uo_j| \neq 0$, or (b) $|\mathbf{x}.xo_j - \mathbf{x}.uo_j| > \beta$ and $\operatorname{sgn}(\mathbf{x}.x_j - \mathbf{x}.xo_j) \neq$ $\operatorname{sgn}(\mathbf{x}.uo_j - \mathbf{x}.xo_j)$. Then, $\mathbf{x} \overset{\mathsf{suspect_i(j)}}{\rightarrow} \mathbf{x}'$.*

Now we show an upper-bound on the number of rounds to detect any failure which may be detected using the failure detection mechanism incorporated in Figure 3 by applying Lemma 8 with Lemmata 9 and 10, and that agents share suspected sets in Figure 3, Line 21. This states an $O(N)$ upper-bound on the detection time of our failure detector and shows that eventually all non-faulty agents know the set of failed agents.

Corollary 2. *For any state $\mathbf{x}_k \in \alpha_{ff}$ such that $\mathbf{x}_k \notin Terminal$, there exists a round $\mathbf{x}_s$ in α_{ff} such that $\forall i \in \bar{F}(\mathbf{x}_s)$, $\mathbf{x}_s.Suspected_i = F(\mathbf{x})$ and $k - s$ is $O(N)$.*

3.4 Simulations

Simulation studies were performed, where flocking convergence time (as by Lemma 4), goal convergence time (as by Theorem 2), and failure detection time (as by Corollary 2) were of interest. Unless otherwise noted, the parameters are chosen as $N = 6$, $N_L = 2$, $r_s = 20$, $r_f = 40$, $\delta = 10$, $\beta = \delta/(4N)$, $v_{min} = \frac{\beta}{2}$, $v_{max} = \beta$, the head agent starts with position at r_f, and the goal is chosen as the origin. Figure 4 shows the value of the Lyapunov function V and maximum agent error from flocking, e_{max}. The initial state is that each agent is spaced by r_s from its left neighbor. Observe that while moving towards the goal, $Flock_S$ is repeatedly satisfied and violated, with invariance of $Flock_W$.

Figure 5 shows that for a fixed value of v_{min}, the time to convergence to NBM is linear in the number of agents. This choice of fixed v_{min} must be for the largest number of agents, 12 in this case, as v_{min} is upper bounded by $\beta = \frac{\delta}{4N}$ which is a function of N. As v_{min} is varied the inverse relationship with N is observed, resulting in a roughly quadratic growth of convergence time to NBM. This illustrates linear convergence time as well as linear detection time, as this is bounded by the convergence time from Corollary 2. The initial state was for expansion, so each agent was spaced at r_s from its left neighbor.

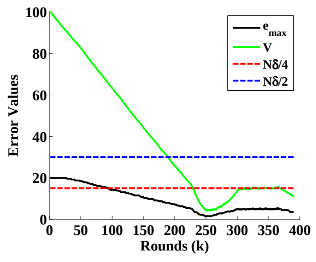

Fig. 4. Expansion simulation showing max error e_{max}, Lyapunov function value V, with weak and strong flocking constants.

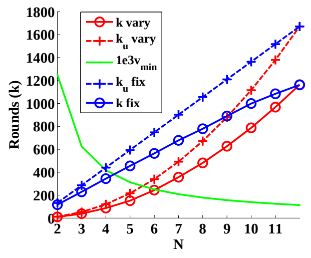

Fig. 5. Rounds k (upper bounded by k_u) to reach $Terminal$ versus number of agents N with fixed and varying v_{min}.

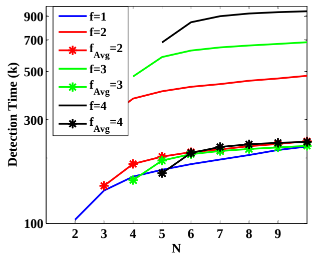

Fig. 6. Multiple failure simulation with f zero velocity failures at round 0 from initial state of $2r_f$ inter-agent spacing.

In all single-failure simulations, a trend was observed on the detection time. When failing each agent individually, and with all else held constant (initial conditions, round of failure, etc.), only one of the detection times for failure velocities of $-v_{max}$, 0, or v_{max} is ever larger than one round. The frequent occurrence of a single round detection is interesting. For instance, in the expansion case, each failed agent i except the tail are detected in one round when $vf_i \neq 0$ since a violation of safety occurs. However, detecting that the head agent has failed with zero velocity requires convergence of the system to a strong flock prior to detection, as does detecting that the tail agent failed with v_{max}, as this mimics the desired expansive behavior up to the point where the tail moves beyond the flock. In the contraction case, each failed agent i except the tail is detected in one round when $vf_i \neq 0$, since they are at the center of their neighbors positions, while the tail agent failing with $-v_{max}$ takes many rounds to detect, since it should be moving towards its left neighbor to cause the contraction. Thus the observation is, for a reachable state $\mathbf{x}$, if $|F(\mathbf{x})| = 1$, let the identifier of the failed agent be i, and consider the three possibilities of $\mathbf{x}.vf_i = 0$, $\mathbf{x}.vf_i \in (0, v_{max}]$, and $\mathbf{x}.vf_i \in [-v_{max}, 0)$. Then along a fail-free execution fragment starting from $\mathbf{x}$, for one of these choices of vf_i, the detection time is greater than 1, and for the other two, the detection time is 1. This illustrates there is only one potentially "bad" mimicking action which allows maintenance of both safety and progress and takes more than one round to detect. The other two failure velocity conditions violate either progress or safety immediately and lead to an immediate detection.

Finally, Figure 6 shows the detection time with varying N and f from a fixed initial condition of inter-agent spacings at $2r_f$. The $f_{Avg} = i$ lines show the total detection time divided by f. Failures were fixed with $vf_i = 0$, failing each combination of agents, so for $f = 2$ and $N = 3$, each combination of $\{1, 2\}$, $\{1, 3\}$, $\{2, 3\}$ were failed individually, and the detection time is the average over the number of these combinations for each choice of f and N. The detection time averaged over the number of failure indicates that the detection time to detect any failure in a multiple failure scenario is on the same order as that in the single failure case. However, the detection time not averaged over the number of failures indicates that the detection time to detect all failures increases linearly in f and on the order of N, as predicated by Corollary 2.

4 Conclusion

This paper presented an algorithm for the safe flocking problem—where the desired properties are safety invariance and eventual progress, that eventually a strong flock is formed and a destination reached by that flock—in spite of permanent actuator faults. An $O(N)$ lower-bound was presented for the detection time of actuator faults, as well as conditions under which the given failure detector can match this bound, although it was established that this is not always possible. The main result was that the algorithm is self-stabilizing when combined with a failure detector. Without the failure detector, the system would not be able to maintain safety as agents could collide, nor make progress to states satisfying flocking or the destination, since failed agents may diverge, causing their neighbors to follow and diverge as well. Simulation results served to reiterate the formal analysis, and demonstrated the influence of certain factors—such as multiple failures—on the failure detection time.

References

1. Arora, A., Gouda, M.: Closure and convergence: A foundation of fault-tolerant computing. IEEE Trans. Softw. Eng. 19, 1015–1027 (1993)
2. Chandra, T.D., Toueg, S.: Unreliable failure detectors for reliable distributed systems. J. ACM 43(2), 225–267 (1996)
3. Chandy, K.M., Lamport, L.: Distributed snapshots: determining global states of distributed systems. ACM Trans. Comput. Syst. 3(1), 63–75 (1985)
4. Dolev, S.: Self-stabilization. MIT Press, Cambridge, MA (2000)
5. Fax, J., Murray, R.: Information flow and cooperative control of vehicle formations. IEEE Trans. Autom. Control 49(9), 1465–1476 (Sep 2004)
6. Franceschelli, M., Egerstedt, M., Giua, A.: Motion probes for fault detection and recovery in networked control systems. In: American Control Conference, 2008. pp. 4358–4363 (Jun 2008)
7. Gazi, V., Passino, K.M.: Stability of a one-dimensional discrete-time asynchronous swarm. IEEE Trans. Syst., Man, Cybern. B 35(4), 834–841 (Aug 2005)
8. Jadbabaie, A., Lin, J., Morse, A.: Coordination of groups of mobile autonomous agents using nearest neighbor rules. IEEE Trans. Autom. Control 48(6), 988–1001 (Jun 2003)
9. Johnson, T.: Fault-Tolerant Distributed Cyber-Physical Systems: Two Case Studies. Master's thesis, Department of Electrical and Computer Engineering, University of Illinois at Urbana-Champaign, Urbana, IL 61801 (May 2010)
10. Johnson, T., Mitra, S.: Safe and stabilizing distributed flocking in spite of actuator faults. Tech. Rep. UILU-ENG-10-2204 (CRHC-10-02), University of Illinois at Urbana-Champaign, Urbana, IL (May 2010)
11. Okubo, A.: Dynamical aspects of animal grouping: Swarms, schools, flocks, and herds. Adv. Biophys. 22, 1–94 (1986)
12. Olfati-Saber, R.: Flocking for multi-agent dynamic systems: algorithms and theory. IEEE Trans. Autom. Control 51(3), 401–420 (Mar 2006)
13. Shaw, E.: Fish in schools. Natural History 84(8), 40–45 (1975)
14. Tsitsiklis, J., Bertsekas, D., Athans, M.: Distributed asynchronous deterministic and stochastic gradient optimization algorithms. IEEE Trans. Autom. Control 31(9), 803–812 (Sep 1986)