JAIST Repository
https://dspace.jaist.ac.jp/

Title	Fault-tolerant Flocking in a k-bounded Asynchronous System
Author(s)	Souissi, Samia; Yang, Yan; Defago, Xavier
Citation	Research report (School of Information Science, Japan Advanced Institute of Science and Technology), IS-RR-2008-004: 1-22
Issue Date	2008-09-26
Type	Technical Report
Text version	publisher
URL	http://hdl.handle.net/10119/4800
Rights	
Description	リサーチレポート（北陸先端科学技術大学院大学情報科学研究科）

JAIST
JAPAN
ADVANCED INSTITUTE OF
SCIENCE AND TECHNOLOGY

Japan Advanced Institute of Science and Technology

Fault-tolerant Flocking in a *k*-bounded Asynchronous System

Samia Souissi, Yan Yang, Xavier Défago

School of Information Science,
Japan Advanced Institute of Science and Technology (JAIST)

September 26, 2008

IS-RR-2008-004

Japan Advanced Institute of Science and Technology (JAIST)

School of Information Science
1-1 Asahidai, Nomi, Ishikawa 923-1292, Japan

http://www.jaist.ac.jp/

ISSN 0918-7553

Fault-tolerant Flocking in a k-bounded Asynchronous System ⋆

Samia Souissi, Yan Yang, and Xavier Défago

School of Information Science
Japan Advanced Institute of Science and Technology (JAIST)
1-1 Asahidai, Nomi, Ishikawa 923-1292, Japan.
{ssouissi,y.yang,defago}@jaist.ac.jp

Abstract. This paper studies the flocking problem, where mobile robots group to form a desired pattern and move together while maintaining that formation. Unlike previous studies of the problem, we consider a system of mobile robots in which a number of them may possibly fail by crashing. Our algorithm ensures that the crash of faulty robots does not bring the formation to a permanent stop, and that the correct robots are thus eventually allowed to reorganize and continue moving together. Furthermore, the algorithm makes no assumption on the relative speeds at which the robots can move.
The algorithm relies on the assumption that robots' activations follow a k-bounded asynchronous scheduler, in the sense that the beginning and end of activations are not synchronized across robots (asynchronous), and that while the slowest robot is activated once, the fastest robot is activated at most k times (k-bounded).
The proposed algorithm is made of three parts. First, appropriate restrictions on the movements of the robots make it possible to agree on a common ranking of the robots. Second, based on the ranking and the k-bounded scheduler, robots can eventually detect any robot that has crashed, and thus trigger a reorganization of the robots. Finally, the third part of the algorithm ensures that the robots move together while keeping an approximation of a regular polygon, while also ensuring the necessary restrictions on their movement.

1 Introduction

Be it on earth, in space, or on other planets, robots and other kinds of automatic systems provide essential support in otherwise adverse and hazardous environments. For instance, among many other applications, it is becoming increasingly attractive to consider a group of mobile robots as a way to provide support for rescue and relief during or after a natural catastrophe (e.g., earthquake, tsunami, cyclone, volcano eruption). As a result, research on mechanisms for coordination and self-organization of mobile robot systems is beginning to attract considerable attention (e.g, [17–20]). For such operations, relying on a group of simple robots for delicate operations has various advantages over considering a single complex robot. For instance, (1) it is usually more cost-effective to manufacture and deploy a number of cheap robots rather than a single expensive one, (2) higher number yields better potential for a system resilient to

⋆ Work supported by MEXT Grant-in-Aid for Young Scientists (A) (Nr. 18680007).

individual robot failures, (3) smaller robots have obviously better mobility in tight and confined spaces, and (4) a group can survey a larger area than an individual robot, even if the latter is equipped with better sensors.

Nevertheless, merely bringing robots together is by no means sufficient, and adequate coordination mechanisms must be designed to ensure coherent group behavior. Furthermore, since many applications of cooperative robotics consider cheap robots dwelling in hazardous environments, fault-tolerance is of primary concern.

The problem of reaching agreement among a group of autonomous mobile robots has attracted considerable attention over the last few years. While much formal work focuses on the gathering problem (robots must meet at a point, e.g., [7]) as the embodiment of a *static* notion of agreement, this work studies the problem of flocking (robots must move together), which embodies a *dynamic* notion of agreement, as well as coordination and synchronization. The flocking problem has been studied from various perspectives. Studies can be found in different disciplines, from artificial intelligence to engineering [1, 3, 5, 6]. However, only few works considered the presence of faulty robots [2, 4].

Fault-tolerant flocking. Briefly, the main problem studied in this paper, namely the flocking problem, requires that a group of robots move together, staying close to each other, and keeping some desired formation while moving. Numerous definitions of flocking can be found in the literature [3, 11, 12, 14], but few of them define the problem precisely. The rare rigorous definitions of the problem suppose the existence of a leader robot and require that the other robots, called followers, follow the leader in a desired fashion [3, 6, 10], such as by maintaining an approximation of a regular polygon.

The variant of the problem that we consider in this paper requires that the robots form and move while maintaining an approximation of a regular polygon, in spite of the possible presence of faulty robots—robots may fail by crashing and a crash is permanent. Although we do consider the presence of a leader robot to lead the group, the role of leader is assigned *dynamically* and any of the robots can potentially become a leader. In particular, after the crash of a leader, a new leader must eventually take over that role.

Model. The system is modelled as a system composed of a group of autonomous mobile robots, modelled as points evolving on the plane, and all of which execute the same algorithm independently. Some of the robots may possibly fail by crashing, after which they do not move forever. Although the robots share no common origin, they do share one common direction (as given by a compass), a common unit distance, and the same notion of clockwise direction.

Robots repeatedly go through a succession of activation cycles during which they observe their environment, compute a destination and move. Robots are asynchronous in that one robot may begin an activation cycle while another robot finishes one. While some robots may be activated more often than others, we assume that the scheduler is k-bounded in the sense that, in the interval it takes any correct robot to perform a single activation cycle, no other robot performs more than k activations. The robots can remember only a *limited* number of their past activations.

Contribution. The paper presents a fault-tolerant flocking algorithm for a k-bounded asynchronous robot system. The algorithm is decomposed into three parts. In the first part, the algorithm relies on the k-bounded scheduler to ensure failure detection. In the second part, the algorithm establishes a ranking system for the robots and then ensures that robots agree on the same ranking throughout activations. In the third and last part, the ranking and the failure detector are combined to realize the flocking of the robots by maintaining an approximation of a regular polygon while moving.

Related work. Gervasi and Prencipe [3] have proposed a flocking algorithm for robots based on a leader-followers model, but introduce additional assumptions on the speed of the robots. In particular, they proposed a flocking algorithm for formations that are symmetric with respect to the leader's movement, without agreement on a common coordinate system (except for the unit distance). However, their algorithm requires that the leader is distinguished from the robots followers.

Canepa and Potop-Butucaru [6] proposed a flocking algorithm in an asynchronous system with oblivious robots. First, the robots elect a leader using a probabilistic algorithm. After that, the robots position themselves according to a specific formation. Finally, the formation moves ahead. Their algorithm only lets the formation move straight forward. Although the leader is determined dynamically, once elected it can no longer change. In the absence of faulty robots, this is a reasonable limitation in their model.

To the best of our knowledge, our work is the first to consider flocking of asynchronous (k-bounded) robots in the presence of faulty robots. Also, we want to stress that the above two algorithms do not work properly in the presence of faulty robots, and that their adaptation is not straightforward.

Structure. The remainder of this paper is organized as follows. In Section 2, we present the system model. In Section 3, we define the problem. In Section 4, we propose a failure detection algorithm based on $k-$bounded scheduler. In Section 5, we give an algorithm that provides a ranking mechanism for robots. In Section 6, we propose a dynamic fault tolerant flocking algorithm that maintains an approximation of a regular polygon. Finally, in Section 7, we conclude the paper.

2 System Model and Definitions

2.1 The CORDA model

In this paper, we consider the CORDA model of Prencipe [8] with k-bounded scheduler. The system consists of a set of autonomous mobile robots $\mathcal{R} = \{r_1, \cdots, r_n\}$. A robot is modelled as a unit having computational capabilities, and which can move freely in the two-dimensional plane. Robots are seen as points on the plane. In addition, they are equipped with sensor capabilities to observe the positions of the other robots, and form a local view of the world.

The local view of each robot includes a unit of length, an origin, and the directions and orientations of the two x and y coordinate axes. In particular, we assume that robots have a partial agreement on the local coordinate system. Specifically, they

agree on the orientation and direction of one axis, say y. Also, they agree on the clockwise/counterclokwise direction.

The robots are completely *autonomous*. Moreover, they are *anonymous*, in the sense that they are a priori indistinguishable by appearance. Furthermore, there is no direct means of communication among them.

In the CORDA model, robots are totally *asynchronous*. The cycle of a robot consists of a sequence of events: Wait-Look-Compute-Move.

- *Wait*. A robot is idle. A robot cannot stay permanently idle. At the beginning all robots are in Wait state.
- *Look*. Here, a robot *observes* the world by activating its sensors, which will return a snapshot of the positions of the robots in the system.
- *Compute*. In this event, a robot *performs* a local computation according to its deterministic algorithm. The algorithm is the same for all robots, and the result of the *compute* state is a destination point.
- *Move*. The robot *moves* toward its computed destination. But, the distance it moves is unmeasured; neither infinite, nor infinitesimally small. Hence, the robot can only go towards its goal, but the move can end anywhere before the destination.

In the model, there are two limiting assumptions related to the cycle of a robot.

Assumption 1. *It is assumed that the distance travelled by a robot r in a move is not infinite. Furthermore, it is not infinitesimally small: there exists a constant $\delta_r > 0$, such that, if the target point is closer than δ_r, r will reach it; otherwise, r will move toward it by at least δ_r.*

Assumption 2. *The amount of time required by a robot r to complete a cycle (wait-look-compute-move) is not infinite. Furthermore, it is not infinitesimally small; there exists a constant $\tau_r > 0$, such that the cycle will require at least τ_r time.*

2.2 Assumptions

k-bounded-scheduler. In this paper, we assume the CORDA model with k-bounded scheduler, in order to ensure some fairness of activations among robots. Before we define the k-bounded-scheduler, we give a definition of *full activation cycle* for robots.

Definition 1 (full activation cycle). *A* **full activation cycle** *for any robot r_i is defined as the interval from the event Look (included) to the next instance of the same event Look (excluded).*

Definition 2 (k-bounded-scheduler). *With a* **k-bounded scheduler***, between two consecutive full activation cycles of the same robot r_i, another robot r_j can execute at most k full activation cycles.*

This allows us to establish the following lemma:

Lemma 1. *If a robot r_i is activated $k+1$ times, then all (correct) robots have completed at least one full activation cycle during the same interval.*

Faults. In this paper, we address ***crash failures***. That is, we consider *initial* crash of robots and also the crash of robots *during execution*. That is, a robot may fail by crashing, after which it executes no actions (no movement). A crash is *permanent* in the sense that a faulty robot never recovers. However, it is still physically present in the system, and it is seen by the other non-crashed robots. A robot that is not faulty is called a *correct* robot.

Before we proceed, we give the following notations that will be used throughout this paper. We denote by $\mathcal{R} = \{r_1, \cdots, r_n\}$ the set of all the robots in the system. Given some robot r_i, $r_i(t)$ is the position of r_i at time t. $y(r_i)$ denotes the y coordinate of robot r_i at some time t. Let A and B be two points, with $\overline{AB}$, we will indicate the segment starting at A and terminating at B, and $dist(A, B)$ is the length of such a segment. Given a region $\mathcal{X}$, we denote by $|\mathcal{X}|$, the number of robots in that region at time t. Finally, let S be a set of robots, then $|S|$ indicates the number of robots in S.

3 Problem Definition

Definition 3 (Formation). *A formation $F = Formation(P_1, P_2, ..., P_n)$ is a configuration, with P_1 the leader of the formation, and the remaining points, the followers of the formation. The leader P_1 is not distinct physically from the robot followers.*

In this paper, we assume that the formation F is a regular polygon. We denote by d the length of the polygon edge (known to the robots), and by $\alpha = (n-2)180°/n$ the angle of the polygon, where n is the number of robots in F.

Definition 4 (Approximate Formation). *We say that robots form an approximation of the formation F if each robot r_i is within ϵ_r from its target P_i in F.*

Definition 5 (The Flocking Problem). *Let r_1,...,r_n be a group of robots, whose positions constitute a formation $F = Formation(P_1, P_2, ..., P_n)$. The robots solve the* **Approximate Flocking Problem** *if, starting from any arbitrary formation at time t_0, $\exists t_1 \geq t_0$ such that, $\forall t \geq t_1$ all robots are at a distance of at most ϵ_r from their respective targets P_i in F, and ϵ_r is a small positive value known to all robots.*

4 Perfect Failure Detection

In this section, we give a simple perfect failure detection algorithm for robots based on a $k-$bounded scheduler in the asynchronous model CORDA. The concept of failure detectors was first introduced by Chandra and Toueg [16] in asynchronous systems with crash faults. A perfect failure detector has two properties: *strong completeness*, and *strong accuracy*. Before we proceed to the description of the algorithm, we make the following assumption, which is necessary for the failure detector mechanism to identify *correct* robots and *crashed* ones.

Assumption 3. *We assume that, at each activation of some robot r_i (correct), r_i computes as destination a position that is different from its current position. Also, a robot*

r_i never visits the same location for the last $k+1$ activations of r_i.[1] *Finally, a robot r_i never visits a location that was visited by any other robot r_j during the last $k+1$ activations of r_j.*

Recall that we only consider *permanent* crash failures of robots, and that crashed robots remain physically in the system. Besides, robots are anonymous. Therefore, the problem is how to distinguish faulty robots from correct ones. Algorithm 1 provides a simple perfect failure detection mechanism for the identification of correct robots. The algorithm is based on the fact that a correct robot must change its current position whenever it is activated (Assumption 3), and also relies on the definition of the $k-$bounded scheduler for the activations of robots. So, a robot r_i considers that some robot r_j is faulty if r_i is activated $k+1$ times, while robot r_j is still in the same position. Algorithm 1 gives as output the set of positions of correct robots $S_{correct}$, and uses the following *variables*:

- $S_{PosPrevObser}$: a global variable representing the set of points of the positions of robots in the system in the previous activation of some robot r_i. These points include the positions of correct and faulty robots. $S_{PosPrevObser}$ is initialized to the empty set during the first activation of robot r_i.
- $S_{PosCurrObser}$: the set of points representing the positions of robots (including faulty ones) in the current activation of some robot r_i.
- c_j: a global variable recording how many times robot r_j did not change its position.

Algorithm 1 Perfect Failure Detection (code executed by robot r_i)

Initialization: $S_{PosPrevObser} := \emptyset$; $c_j := 0$

```
procedure Failure_Detection(S_PosPrevObser, S_PosCurrObser)
    S_correct := S_PosCurrObser;
    for ∀ p_j ∈ S_PosCurrObser do
        if (p_j ∈ S_PosPrevObser) then                    {robot r_j has not moved}
            c_j := c_j + 1;
        else
            c_j := 0;
        end if
        if (c_j ≥ k) then
            S_correct = S_correct − {p_j};
        end if
    end for
    return (S_correct)
end
```

The proposed failure detection algorithm (Algorithm 1) satisfies the two properties of a perfect failure detector: *strong completeness*, and *strong accuracy*. It also satisfies

[1] That is, r_i never revisits a point location that was within its line of movement for its last $k+1$ total activations.

the *eventual agreement* property. These properties are stated respectively in Theorem 1, Theorem 2, and Theorem 3, and their proofs are deferred to Appendix A.

Theorem 1. ***Strong completeness****: eventually every robot that crashes is permanently suspected by every correct robot.*

Theorem 2. ***Strong accuracy****: there is a finite time after which correct robots are not suspected by any other correct robots.*

Theorem 3. ***Eventual agreement****: there is a finite time after which, all correct robots agree on the same set of correct robots in the system.*

5 Agreed Ranking for Robots

In this section, we provide an algorithm that gives a unique ranking (or identification) to every robot in the system since we assume that robots are anonymous, and do not have any identifier to allow them to distinguish each other. The algorithm allows correct robots to compute and agree on the same ranking. In particular, the ranking mechanism is needed for the election of the leader of the formation. Recall that, a deterministic leader election is impossible without a shared y-axis [9]. Therefore, we assume that robots agree on the y-axis.

We first assume that robots are not located initially at the same point. That is, robots are not in the gathering configuration [7], because it may become impossible to separate them later.[2] The ranking assignment is given in Algorithm 2, which takes as input the set of positions of correct robots in the system $S_{correct}$, and returns as output an ordered set of the positions in $S_{correct}$, called $RankSequence$. The ranking of positions of robots in $S_{correct}$ gives to every robot a unique identification number. The computation of $RankSequence$ is done as follows: $RankSequence = \{S_{correct}, <\}$, where the relation " $<$ "is defined by comparing the y coordinates of the points in $S_{correct}$, and breaking ties from left to right. In other words, the positions of robots in $S_{correct}$ are sorted by decreasing order of $y-$coordinate, such that the robot with greatest y-coordinate is the first in $RankSequence$. When two or more robots share the same y-coordinate, the clockwise direction is used to determine the sequence; a robot r_i that has a robot r_j on its right hand, has a lower rank than r_j in $RankSequence$.

In order for robots to agree on the same $RankSequence$ initially, some restrictions on their movement are required during their first k activations. The movement restriction is given by procedure Lateral_Move_Right(), and it is designed in such a way that all robots compute the same $RankSequence$ during their first k activations. In particular, a robot r_i that does not have robots on $Right(r_i)$ can move by at most the distance $\epsilon_r/(k+1)(k+2)$ along $Right(r_i)$ in order to preserve the same $y-$coordinate. Otherwise, r_i moves by $min(\epsilon_r/(k+1)(k+2), dist(r_i, p)/(k+1)(k+2))$ along $Right(r_i)$,

[2] Consider two robots that happen to have the same coordinate system and that are always activated together. It is impossible to separate them deterministically. In contrast, it would be trivial to scatter them at distinct positions using randomization (e.g., [15]), but this is ruled out in our model.

Algorithm 2 Ranking_Correct_Robots (code executed by robot r_i)

Input: $S_{correct}$: set of positions of correct robots;
Output: $RankSequence$: Ordered set of positions of correct robots $S_{correct}$;
Initialization: $counter_{act}$:= a global variable recording the number of activations of robot r_i;
procedure Ranking_Correct_Robots($S_{correct}$)
  When r_i **is activated**
  $counter_{act} := counter_{act} + 1$;
  $Left(r_i)$:= is the ray starting at r_i and perpendicular to its $y-$axis in counter-clockwise direction.
  Sort the $y-$coordinates of robots in $S_{correct}$ in decreasing order.
  if $(\forall r_j, r_k \in S_{correct}, y(r_j) \neq y(r_k))$ **then**
    $RankSequence$:= the set $S_{correct}$ in order of decreasing $y-$coordinate;
  else if $y(r_j) = y(r_k)$ **then**
    if (r_j is on $Left(r_k)$) **then**
      $RankSequence := r_j < r_k$;
    else
      $RankSequence := r_k < r_j$;
    end if
  end if
  if $(counter_{act} \leq k)$ **then**
    Lateral_Move_Right();
  end if
  Return($RankSequence$);
end

Algorithm 3 Procedure Lateral Move Right (code executed by robot r_i).

procedure Lateral_Move_Right()
  $Right(r_i)$:= the ray starting at r_i and perpendicular to its $y-$axis in clockwise direction;
  if (If no other robot on $Right(r_i)$) **then**
    r_i moves by at most $\epsilon_r/(k+1)(k+2)$ to $Right(r_i)$;
  else {*some robots are in* $Right(r_i)$ *including faulty robots*}
    p := the position of the nearest robot to r_i in $Right(r_i)$;
    r_i moves by $min(\epsilon_r/(k+1)(k+2), dist(r_i, p)/(k+1)(k+2))$ to $Right(r_i)$;
  end if
end

where p is the position of the nearest robot to r_i in $Right(r_i)$. [3] From Algorithm 2, we derive the following lemmas. In particular, the algorithm gives a unique ranking to every robot in the system, and also ensures no collisions between robots.

Lemma 2. *Algorithm 2 gives a unique ranking to every correct robot in the system.*

Lemma 3. *By Algorithm 2, there is a finite time after which, all correct robots agree on the same initial sequence of ranking, $RankSequence$.*

Lemma 4. *Algorithm 2 guarantees no collisions between the robots in the system.*

The proofs of the above lemmas are straightforward, and thus given in Appendix B.

6 Dynamic Fault-tolerant Flocking

In this section, we propose a dynamic fault tolerant flocking algorithm, where a group of robots can dynamically generate an approximation of a regular polygon (Definition 4), and maintain it while moving. Our flocking algorithm relies on the existence of two devices, namely a *perfect failure detector* device and a *ranking* device, which were represented respectively in Algorithm 1, and Algorithm 2.

6.1 Algorithm Description

The flocking algorithm is depicted in Algorithm 4, and takes as *input* the length of the polygon edge d, and the *history* of robot r_i, which includes the following variables:

- $S_{PosPrevObser}$: the set of positions of robots in the system during the last previous observation of robot r_i.
- $HistoryMove$: the set of points on the plane visited by robot r_i during its last previous $k+1$ activations.
- nbr_{act}: a counter recording the last previous $k+1$ activations of robot r_i.

The overall idea of the algorithm is as follows. First, when robot r_i gets activated, it executes the following steps:

1. Robot r_i takes a snapshot of the current positions $S_{PosCurrObser}$ of robots in the system.
2. Robot r_i calls the failure detection module to get the set of correct robots, $S_{correct}$.
3. Robot r_i calls the ranking module, and gets a total ordering on the set of correct robots $S_{correct}$, called $RankSequence$.
4. Depending on the rank of robot r_i in $RankSequence$, r_i executes the procedure described in Algorithm 5; $\mathsf{Flocking_Leader}(\mathsf{RankSequence}, \mathsf{d}, \mathsf{nbr_{act}}, \mathsf{HistoryMove})$ if it has the first rank in $RankSequence$ (i.e., the leader). Otherwise, robot r_i is a follower, and it executes the procedure which is described in Algorithm 6, $\mathsf{Flocking_Follower}(\mathsf{RankSequence}, \mathsf{d}, \mathsf{nbr_{act}}, \mathsf{HistoryMove})$.

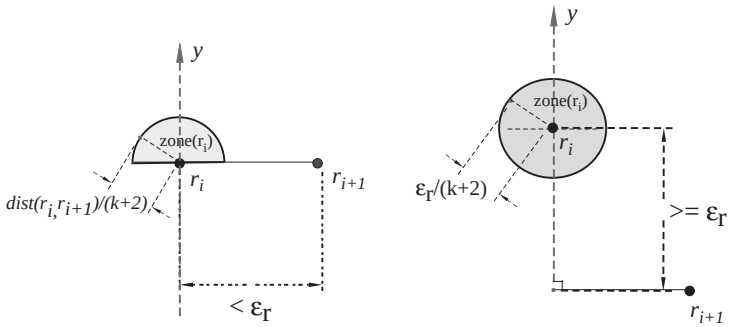
[3] Note that, the bounded distance $min(\epsilon_r/(k+1)(k+2), dist(r_i,p)/(k+1)(k+2))$ set on the movement of robots is conservative, and is sufficient to avoid collisions between robots, and to satisfy Assumption 3.

Algorithm 4 Dynamic Fault-tolerant Flocking (code executed by robot r_i)

Input: Memory(r_i):$S_{PosPrevObser}$;$HistoryMove$;nbr_{act};
d = the desired distance of the polygon edge;
When r_i **is activated**
r_i takes a snapshot of the positions $S_{PosCurrObser}$ of robots;
$S_{correct}$ = Failure_Detection($S_{PosPrevObser}$, $S_{PosCurrObser}$);
$RankSequence$ = Ranking_Correct_Robots($S_{correct}$);
$leader$:= first robot in $RankSequence$;
if ($r_i = leader$) **then** $\{leader\}$
  Flocking_Leader(RankSequence, d, nbr_{act}, HistoryMove);
else $\{follower\}$
  Flocking_Follower(RankSequence, d, nbr_{act}, HistoryMove);
end if

Algorithm 5 Flocking Leader: Code executed by a robot leader r_i.

procedure Flocking_Leader($RankSequence$,d,nbr_{act}, $HistoryMove$)
  $n := |RankSequence|$;
  $\alpha := (n-2)180°/n$;
  P := Formation($P_1, P_2, ..., P_n$) as in Definition 3;
  P_1 := current position of the leader r_i;
  r_{i+1}:= next robot to r_i in $RankSequence$;
  $proj_{r_{i+1}}$:= the projection of r_{i+1} on $y-$axis of r_i;
  if ($proj_{r_{i+1}} = r_i$) **then** $\{r_i$ *has same* $y-$*coordinate as* $r_{i+1}\}$
    $Zone(r_i)$:= half circle with radius $min(dist(r_i, r_{i+1})/(k+1)(k+2), \epsilon_r/(k+1)(k+2))$, centered at r_i and above r_i (refer to Fig. 1(a));
  else
    $Zone(r_i)$:= the circle centered at r_i, and with radius $min(\epsilon_r/(k+1)(k+2), dist(r_i, proj_{r_{i+1}})/(k+1)(k+2))$ (refer to Fig. 1(b));
  end if
  $S_{CrashInZone}$:= the set of positions of crashed robots in $Zone(r_i)$;
  if ($S_{CrashInZone} \neq \emptyset$) **then**
    r_i moves to a desired point $Target(r_i)$ within $Zone(r_i)$, excluding the points in $S_{CrashInZone}$, and the points in $HistoryMove$;
  else
    r_i moves to a desired point $Target(r_i)$ within $Zone(r_i)$, excluding the points in $HistoryMove$;
  end if
  $CurrMove$:= the set of points on the segment $\overline{r_i Target(r_i)}$;
  if ($nbr_{act} \leq k+1$) **then**
    $HistoryMove := HistoryMove \cup CurrMove$;
  else
    $HistoryMove := CurrMove$;
    $nbr_{act} := 1$;
  end if
end



(a) r_i and r_{i+1} have the same y-coordinate, and $dist(r_i, r_{i+1}) < \epsilon_r$: $Zone(r_i)$ is the half circle with radius $dist(r_i, r_{i+1})/(k+1)(k+2)$.

(b) r_i and r_{i+1} do not have the same y-coordinate, and $dist(r_i, proj_{r_{i+1}}) \geq \epsilon_r$: $Zone(r_i)$ is the circle with radius $\epsilon_r/(k+1)(k+2)$.

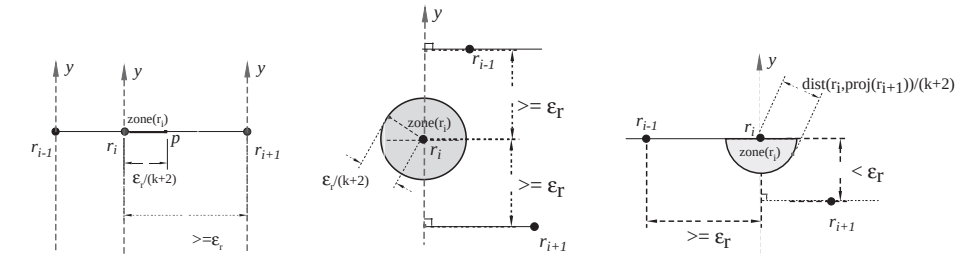
Fig. 1. Zone of movement of the leader.

5. Robot r_i is a *leader*. First, r_i computes the points of the formation $P_1, ..., P_n$ as in Definition 4, with its location as the first point P_1 in the formation. The targets of the followers are the other points of the formation, and they are assigned to them based on their order in the $RankSequence$. After that, the leader will initiate the movement of the formation, while preserving the same rank sequence, keeping an approximation of the regular polygon, and also avoiding collisions with followers.

   In order to prevent collisions between robots, the algorithm must guarantee that no two robots ever move to the same location. Therefore, the algorithm defines a movement zone for each robot, within which the robot must move. The zone of the leader, referred to as $Zone(r_i)$, is defined depending on the position of the next robot r_{i+1} in $RankSequence$. Let us denote by $proj_{r_{i+1}}$, the projection of robot r_{i+1} on the $y-$axis of r_i. The movement zone of the leader is defined as follows:

   – r_i and r_{i+1} have the same y coordinate: $Zone(r_i)$ is the half circle with radius $min(dist(r_i, r_{i+1})/(k+1)(k+2), \epsilon_r/(k+1)(k+2))$, centered at r_i and above r_i (refer to Fig. 1(a)).
   – r_i and r_{i+1} do not have the same y coordinate: $Zone(r_i)$ is the circle, centered at r_i, and with radius $min(dist(r_i, proj_{r_{i+1}})/(k+1)(k+2), \epsilon_r/(k+1)(k+2))$ (refer to Fig. 1(b)).

   After determining its zone of movement $Zone(r_i)$, robot r_i needs to determine if there are crashed robots within $Zone(r_i)$. If no crashed robots are within its zone, then robot r_i can move to any desired target within $Zone(r_i)$, satisfying Assumption 3. Otherwise, robot r_i can move within $Zone(r_i)$ by excluding the positions of crashed robots, and satisfying Assumption 3.
6. Robot r_i is a *follower*. First, r_i assigns the points of the formation $P_1, ..., P_n$ to the robots in $RankSequence$ based on their order in $RankSequence$. Subsequently, robot r_i determines its target P_i based on the current position of the leader (P_1),



(a) r_{i-1}, r_i, and r_{i+1} have the same y coordinate.

(b) r_{i-1}, r_i and r_{i+1} do not have the same y coordinate, and $dist(r_i, proj_{r_{i-1}}) \geq \epsilon_r$, and $dist(r_i, proj_{r_{i+1}}) \geq \epsilon_r$.

(c) r_{i-1} and r_i have the same y coordinate, however, r_{i+1} does not. Also, $dist(r_i, r_{i-1}) \geq \epsilon_r$, and $dist(r_i, proj_{r_{i+1}}) < \epsilon_r$

Fig. 2. Zone of movement of a follower.

and the polygon angle α given in the following equation: $\alpha = (n-2)180°/n$, where n is the number of robots in the formation.

In order to ensure no collisions between robots, the algorithm also defines a movement zone for each robot follower. The zone of a follower, referred to as $Zone(r_i)$ is defined depending on the position of the previous robot r_{i-1} and the next robot r_{i+1} to r_i in $RankSequence$. Before we proceed, we denote by $proj_{r_{i-1}}$, the projection of robot r_{i-1} on the $y-$axis of robot r_i. Similarly, we denote by $proj_{r_{i+1}}$, the projection of robot r_{i+1} on the $y-$axis of r_i. The zone of movement of a robot follower r_i is defined as follows:

- r_i, r_{i-1} and r_{i+1} have the same y coordinate, then $Zone(r_i)$ is the segment $\overline{r_i p}$, with p as the point at distance $min(dist(r_i, r_{i+1})/(k+1)(k+2), \epsilon_r/(k+1)(k+2))$ from r_i (see Fig. 2(a)).
- r_i, r_{i-1} and r_{i+1} do not have the same y coordinate, then $Zone(r_i)$ is the circle centered at r_i, and with radius $min(\epsilon_r/(k+1)(k+2), dist(r_i, proj_{r_{i-1}})/(k+1)(k+2), dist(r_i, proj_{r_{i+1}})/(k+1)(k+2))$ (see Fig. 2(b)).
- r_i and r_{i+1} have the same y coordinate, however r_{i-1} does not, then $Zone(r_i)$ is the half circle above it, centered at r_i, and with radius $min(\epsilon_r/(k+1)(k+2), dist(r_i, proj_{r_{i-1}})/(k+1)(k+2), dist(r_i, r_{i+1})/(k+1)(k+2))$.
- r_i and r_{i-1} have the same y coordinate, however r_{i+1} does not, then $Zone(r_i)$ is the half circle below it, centered at r_i, and with radius $min(\epsilon_r/(k+1)(k+2), dist(r_i, r_{i-1})/(k+1)(k+2), dist(r_i, proj_{r_{i+1}})/(k+1)(k+2))$ (see Fig. 2(c)).

As we mentioned before, the bounded distance $min(\epsilon_r/(k+1)(k+2), dist(r_i, p)/(k+1)(k+2))$ set on the movement of robots is conservative, and is sufficient to avoid collisions between robots, and to satisfy Assumption 3 (this will be proved later).

For the sake of clarity, we do not describe explicitly in Algorithm 6 the zone of movement of the last robot in the rank sequence. The computation of its zone of movement is similar to that of the other robot followers, with the only difference being that it does not have a next neighbor r_{i+1}. So, if robot r_i has the

same $y-$coordinate as its previous neighbor r_{i-1}, then its zone of movement is the half circle with radius $min(\epsilon_r/(k+1)(k+2), dist(r_i, r_{i-1})/(k+1)(k+2))$, centered at r_i and below r_i. Otherwise, the circle centered at r_i, and with radius $min(\epsilon_r/(k+1)(k+2), dist(r_i, proj_{r_{i-1}})/(k+1)(k+2))$.
After determining its zone of movement $Zone(r_i)$, robot r_i needs to determine if it can progress toward its target $Target(r_i)$. Note that, $Target(r_i)$ may not necessarily belong to $Zone(r_i)$. To do so, robot r_i computes the intersection of the segment $\overline{r_i Target(r_i)}$ and $Zone(r_i)$, called $Intersect$. If $Intersect$ is equal to the position of r_i, then r_i will move toward its right as given by the procedure Lateral_Move_Right(). Otherwise, r_i moves along the segment $Intersect$ as much as possible, while avoiding to reach the location of a crashed robot in $Intersect$, if any, and satisfying Assumption 3. In any case, if r_i is not able to move to any point in $Intersect$, except its current position, it moves to its right as in the procedure Lateral_Move_Right().

Note that, by the algorithm robot followers can move in any direction by adaptation of their target positions with respect to the new position of the leader. When the leader is idle, robot followers move within the distance $\epsilon_r/(k+1)(k+2)$ or smaller in order to keep an approximation of the formation with respect to the position of the leader, and preserve the rank sequence.

6.2 Correctness of the Algorithm

In this section, we prove the correctness of our flocking algorithm by first showing that correct robots agree on the same ranking during the execution of Algorithm 4 (Theorem 4). Second, we prove that no two correct robots ever move to the same location, and that a correct robot never moves to a location occupied by a faulty robot (Theorem 5). Then, we show that all correct robots dynamically form an approximation of a regular polygon in finite time, and keep this formation while moving (Theorem 6). Finally, we prove that our algorithm tolerates permanent failures of robots (Theorem 7).

Lemma 5. *Algorithm 4 satisfies Assumption 3.*

Proof (Lemma 5). To prove the lemma, we first show that any robot r_i in the system is able to move to a destination that is different from its current location, and robot r_i never visits a point location that was within its line of movement for its last $k+1$ activations. Then, we show that a robot r_i never visits a location that was visited by another robot r_j during the last $k+1$ activations of r_j.

First, assume that robot r_i is the leader. By Algorithm 4, its zone of movement $Zone(r_i)$ is either a circle or a half circle on the plane, excluding the points in its history of moves $HistoryMove$ for the last $k+1$ activations, and the positions of crashed robots. Since, $Zone(r_i)$ is composed of an infinite number of points, the positions of crashed robots are finite, and $HistoryMove$ is a strict subset of $Zone(r_i)$, then robot r_i can always compute and move to a new location that is different from the locations visited by r_i during its last $k+1$ activations.

Now, assume that robot r_i is a follower, and let r_{i-1} and r_{i+1}, be respectively the previous, and next robots to r_i in $RankSequence$. Two cases follow depending on the zone of movement of r_i.

Algorithm 6 Flocking Follower: Code executed by a robot follower r_i.

```
procedure Flocking_Follower(RankSequence,d,nbr_act,HistoryMove)
  n := |RankSequence|;
  α := (n − 2)180°/n;
  P := Formation(P_1, P_2, ..., P_n) as in Definition 3;
  P_1 := current position of the leader;
  ∀r_j ∈ RankSequence, Target(r_j) = P_j ∈ Formation(P_1, P_2, ..., P_n);
  if (∀r_j ∈ RankSequence, r_j is within ε_r of P_j) then        {Formation = True}
    Lateral_Move_Right();
  else                                                {Flocking and formation generation}
    r_{i−1}:= previous robot to r_i in RankSequence;
    proj_{r_{i−1}}:= the projection of r_{i−1} on y−axis of r_i;
    r_{i+1} := next robot to r_i in RankSequence;
    proj_{r_{i+1}} := the projection of r_{i+1} on y−axis of r_i;
    if (proj_{r_{i−1}} = r_i ∧ proj_{r_{i+1}} = r_i) then      {r_i has the same y coordinate as its
    neighbors}
      Zone(r_i) := segment with length min(ε_r/(k+1)(k+2), dist(r_i, r_{i+1})/(k+
      1)(k + 2)) starting at r_i to Right(r_i) (see Fig. 2(a));
    else if (proj_{r_{i−1}} ≠ r_i) ∧ (proj_{r_{i+1}} ≠ r_i) then
      Zone(r_i) := circle centered at r_i, with radius min(ε_r/(k + 1)(k +
      2), dist(r_i, proj_{r_{i−1}})/(k+1)(k+2), dist(r_i, proj_{r_{i+1}})/(k+1)(k+2)) (see
      Fig. 2(b));
    else if (proj_{r_{i−1}} ≠ r_i ∧ proj_{r_{i+1}} = r_i) then
      Zone(r_i) := half circle centered at r_i, with radius min(ε_r/(k + 1)(k +
      2), dist(r_i, proj_{r_{i−1}})/(k + 1)(k + 2), dist(r_i, r_{i+1})/(k + 1)(k + 2)), and
      above r_i;
    else                                          {r_i has different y coordinate from next robot}
      Zone(r_i) := half circle centered at r_i, with radius min(ε_r/(k + 1)(k +
      2), dist(r_i, r_{i−1})/(k + 1)(k + 2), dist(r_i, proj_{r_{i+1}})/(k + 1)(k + 2)), and
      below r_i (see Fig. 2(c));
    end if
    Intersect := the intersection of the segment r_iTarget(r_i) with Zone(r_i);
    if (Intersect ≠ r_i) then                          {r_i is able to progress to its target}
      S_CrashInLine := the set of crashed robots that belongs to the segment
      intersect;
      if (S_CrashInLine = ∅) then
        r_i moves linearly to the last point in Intersect, excluding the points in
        HistoryMove;
      else
        r_c := the closest crashed robot to r_i in Intersect;
        r_i moves linearly to the last point in the segment r_ir_c, excluding the point
        r_c, and the points in HistoryMove;
      end if
    else
      Lateral_Move_Right();
    end if
  end if
  CurrMove := the set of points on the segment r_iTarget(r_i);
  if (nbr_act ≤ k + 1) then
    HistoryMove := HistoryMove ∪ CurrMove;
  else
    HistoryMove := CurrMove;
    nbr_act := 1;
  end if
end
```

- Consider the case where $Zone(r_i)$ is the *segment* with length $min(\epsilon_r/(k+1)(k+2), dist(r_i, r_{i+1})/(k+1)(k+2))$, excluding r_i. Since, such case occurs only when r_{i-1}, r_i, and r_{i+1} have the same y coordinate, and robot r_i is only allowed to move to $Right(r_i)$. Then, r_i can always move to a free position in $Right(r_i)$ that does not belong to $HistoryMove$, and that excludes the positions of crashed robots since they are finite and there exists an infinite number of points in $Zone(r_i)$.
- Consider the case where $Zone(r_i)$ is either a circle or a half circle, centered at r_i and with a radius greater than zero, excluding its history of move $HistoryMove$ for the last $k+1$ activations, and the positions of crashed robots. By similar arguments as above, we have $Zone(r_i)$ is composed of an infinite number of points, $HistoryMove$ is a strict subset of $Zone(r_i)$, and the positions of crashed robots are finite. Thus, robot r_i can always compute and move to a new location that is different from the locations visited by r_i during its last $k+1$ activations.

We now show that robot r_i never visits a location that was visited by another robot r_j during the last previous $k+1$ activations of r_j. Without loss of generality, we consider robot r_i and its next neighbor r_{i+1}. The same proof holds for r_i and its previous neighbor r_{i-1}. Observe that if r_i and r_{i+1} are moving away from each other, then neither robots move to a location that was occupied by the other one for its last $k+1$ activations.

Now assume that both robots r_i and r_{i+1} are moving to the same direction, then we will show that r_i never reaches the position of r_{i+1} after $k+1$ activations of r_{i+1}. Assume the worst case where robot r_{i+1} is activated once during each k activations of r_i. Then, after $k+1$ activations of r_{i+1}, r_i will move toward r_{i+1} by a distance of at most $dist(r_i, r_{i+1})(k+1)^2/(k+1)(k+2)$, which is strictly less than $dist(r_i, r_{i+1})$, hence r_i is unable to reach the position of r_{i+1}.

Finally, we assume that both r_i and r_{i+1} are moving toward each other. In this case, we assume the worst case when both robots are always activated together. After $k+1$ activations of either r_i or r_{i+1}, each of them will travel toward the other one by at most the distance $dist(r_i, r_{i+1})(k+1)/(k+1)(k+2)$. Consequently, $2dist(r_i, r_{i+1})/(k+2)$ is always strictly less than $dist(r_i, r_{i+1})$ because $k \geq 1$. Hence, neither r_i or r_{i+1} moves to a location that was occupied by the other during its last $k+1$ activations, and the lemma holds. □

Corollary 1. *By Algorithm 4, at any time t, there is no overlap between the zones of movement of any two correct robots in the system.*

Agreement on Ranking. In this section, we show that correct robots agree always on the same sequence of ranking even in the presence of failure of robots.

Lemma 6. *By Algorithm 4, correct robots always agree on the same $RankSequence$ when there is no crash. Moreover, if some robot r_j crashes, there is a finite time after which, all correct robots exclude r_j from the ordered set $RankSequence$, and keep the same total order in $RankSequence$.*

Proof (Lemma 6). By Lemma 3, all correct robots agree on the same sequence of ranking, $RankSequence$ after the first k activations of any robot in the system. Then, in

the following, we first show that the $RankSequence$ is preserved during the execution of Algorithm 4 when there is no crash in the system. Second, we show that if some robot r_j has crashed, there is a finite time after which correct robots agree on the new sequence of ranking, excluding r_j.

- There is *no crash* in the system: we consider three consecutive robots r_a, r_b and r_c in $RankSequence$, such that $r_a < r_b < r_c$. We prove that the movement of r_b does not allow it to swap ranks with r_a or r_c in the three different cases that follow:
  1. r_a, r_b and r_c share the same y coordinate. In this case, r_b moves by $min(\epsilon_r/(k+1)(k+2), dist(r_b, r_c)/(k+1)(k+2))$ along the segment $\overline{r_b r_c}$. Such a move does not change the y coordinate of r_b, and also it does not change its rank with respect to r_a and r_c because it always stays between r_a and r_c, and it never reaches either r_a nor r_b, by the restrictions on the algorithm.
  2. r_a, r_b and r_c do not share the same y coordinate. In this case, the movement of r_b is restricted within a circle $\mathcal{C}$, centered at r_b, and having a radius that does not allow r_b to reach the same y coordinate as either r_a nor r_c. In particular, the radius of $\mathcal{C}$ is equal to $min(\epsilon_r/(k+1)(k+2), dist(r_b, proj_{r_a})/(k+1)(k+2), dist(r_b, proj_{r_c})/(k+1)(k+2))$, which is less than $dist(r_b, proj_{r_a})/k$, and $dist(r_b, proj_{r_c})/k$, where $proj_{r_a}$ and $proj_{r_c}$ are respectively, the projections of robot r_a and r_c on the $y-$axis of r_b. Hence, such a restriction on the movement of r_b does not allow it to swap its rank with either r_a or r_b.
  3. Two consecutive robots have the same y coordinate, (say r_a and r_b), however r_c does not. This case is almost similar to the previous one. The movement of r_b is restricted within a half circle, centered at r_b, and below it, and with a radius that does not allow r_b to have less than or equal y coordinate as r_c. In particular, that radius is equal to $min(\epsilon_r/(k+1)(k+2), dist(r_a, r_b)/(k+1)(k+2), dist(r_b, proj_{r_c})/(k+1)(k+2))$, which is less than $dist(r_a, r_b)/k$, and also less than $dist(r_b, proj_{r_c})/k$, where $proj_{r_c}$ is the projection of robot r_c on the $y-$axis of r_b. Hence, the restriction on the movement of r_b does not allow it to swap ranks with either r_a or r_b.

  Since, all robots execute the same algorithm, then the proof holds for any two consecutive robots in $RankSequence$. Note that, the same proof applies for both algorithms executed by the leader and the followers because the restrictions made on their movements are the same
- Some robot r_j *crashes*: From what we proved above, we deduce that all robots agree and preserve the same sequence of ranking, $RankSequence$ in the case of no crash. Assume now that a robot r_j crashes. By Lemma 11, we know that there is a finite time after which all correct robots detect the crash of r_j. Hence, there is a finite time after which correct robots exclude robot r_j from the ordered set $RankSequence$.

In conclusion, the total order in $RankSequence$ is preserved for correct robots during the entire execution of Algorithm 4. This terminates the proof. □

The following Theorem is a direct consequence from Lemma 6.

Theorem 4. *By Algorithm 4, all robots agree on the total order of their ranking during the entire execution of the algorithm.*

Collision-Freedom.

Lemma 7. *Under Algorithm 4, at any time t, no two correct robots ever move to the same location. Also, no correct robot ever moves to a position occupied by a faulty robot.*

Proof (Lemma 7). To prove that no two correct robots ever move to the same location, we show that any robot r_i always moves to a location within its own zone $Zone(r_i)$, and the rest follows from the fact that the zones of two robots do not intersect (Corollary 1). By restriction on the algorithm, r_i must move to a location $Target(r_i)$, which is within $Zone(r_i)$. Since, r_i belongs to $Zone(r_i)$, $Zone(r_i)$ is a convex form or a line segment, and the movement of r_i is linear, so all points between r_i and $Target(r_i)$ must be in $Zone(r_i)$.

Now we prove that, no correct robot ever moves to a position occupied by a crashed robot. By Theorem 1, robot r_i can compute the positions of crashed robots in finite time. Moreover, by Lemma 5, robot r_i always has free destinations within its zone $Zone(r_i)$, which excludes crashed robots. Finally, Algorithm 4 restricts robots from moving to the locations that are occupied by crashed robots. Thus, robot r_i never moves to a location that is occupied by a crashed robot. □

The following theorem is a direct consequence from Lemma 7.

Theorem 5. *Algorithm 4 is collision free.*

Fault-tolerant Flocking. Before we proceed, we state the following lemma, which sets a bound on the number of faulty robots under which a polygon can be formed.

Lemma 8. *A polygon is generated if and only if the number of faulty robots f is bounded by $f \leq n - 3$, where n is the number of robots in the system, and $n \geq 3$.*

Proof (Lemma 8). The proof is trivial. A polygon requires three or more robots to be formed. Then, the number of robots n in the system should be greater or equal to three. Also, the number of faulty robots f at any time t in the system should be less than or equal to $n - 3$ for the polygon to be formed. This proves the lemma. □

Lemma 9. *Algorithm 4 allows correct robots to form an approximation of a regular polygon in finite time, and to maintain it in movement.*

Proof (Lemma 9). We first show that each robot can be within ϵ_r of its target in the formation $F(P_1, P_2, ..., P_n)$ in a finite number of steps. Second, we show that correct robots maintain an approximation of the formation while moving.

Assume that r_i is a correct robot in the system. If r_i is a leader, then by Algorithm 4, the target of r_i is a point within a circle or half circle, centered at r_i, and with radius less than or equal to ϵ_r satisfying Assumption 3, and excluding the positions of crashed robots. Since, there exists an infinite number of points within $Zone(r_i)$, and by Assumption 2, the cycle of a robot is finite, then r_i can reach its target within $Zone(r_i)$ in a finite number of steps.

Now, consider that r_i is a robot follower. We also show that r_i can reach within ϵ_r of its target P_i in a finite number of steps. We consider two cases:

- Robot r_i can move freely toward its target P_i: every time r_i is activated, it can progress by at most $\epsilon_r/(k+1)(k+2)$. Since, the distance $dist(r_i, P_i)$ is finite, the bound k of the scheduler is also finite, and the cycle of a robot is finite by Assumption 2, then r_i can be within ϵ_r of P_i in a finite number of steps.
- Robot r_i cannot move freely toward its target P_i: first, assume that r_i cannot progress toward its target because of the restriction on $RankSequence$. Since, there exists at least one robot in $RankSequence$ that can move freely toward its target, and this is can be done in finite time. In addition, the number of robots in $RankSequence$ is finite, and by Lemma 5, a robot can always move to a new location satisfying Assumption 3, then, eventually each robot r_i in $RankSequence$ can progress toward its target P_i, and arrive within ϵ_r of it in a finite number of steps.

  Now, assume that r_i cannot progress toward its target P_i because it is blocked by some crashed robots. By Lemma 5, a robot can always move to a new location satisfying Assumption 3. Also, the number of crashed robots is finite, so eventually robot r_i can make progress, and be within ϵ_r of its target in a finite number of steps, by similar arguments.

We now show that correct robots maintain an approximation of the formation while moving. Since, all robots are restricted to move within one cycle by at most $\epsilon_r/(k+1)(k+2)$, then in every new k activations in the system, each correct robot r_i cannot go farther away than ϵ_r from its position during k activations. Consequently, r_i can always be within ϵ_r of its target P_i as in Definition 4, and the lemma follows. □

Theorem 6. *Algorithm 4 allows correct robots to dynamically form an approximation of a regular polygon, while avoiding collisions.*

Proof (Theorem 6). First, by Theorem 3, there is a finite time after which all correct robots agree on the same set of correct robots. Second, by Theorem 4, all correct robots agree on the total order of their ranking $RankSequence$. Third, By Theorem 5, there is no collision between any two robots in the system, including crashed ones. Finally, by Lemma 9, all correct robots form an approximation of a regular polygon in finite time, and the theorem holds. □

Lemma 10. *Algorithm 4 tolerates permanent crash failures of robots.*

Proof (Lemma 10). By Theorem 1, a crash of a robot is detected in finite time, and by Algorithm 4, a crashed robot is removed from the list of correct robots, although it appears physically in the system. Finally, by Theorem 5, correct robots avoid collisions with crashed robots. Thus, Algorithm 4 tolerates permanent crash failures of robots. □

From Theorem 6, and Lemma 10, we infer the following theorem:

Theorem 7. *Algorithm 4 is a fault tolerant dynamic flocking algorithm that tolerates permanent crash failures of robots.*

7 Conclusion

In this paper, we have proposed a fault-tolerant flocking algorithm that allows a group of asynchronous robots to self organize dynamically, and form an approximation of a regular polygon, while maintaining this formation in movement. The algorithm relies on the assumption that robots' activations follow a k-bounded asynchronous scheduler, and that robots have a limited memory of the past.

Our flocking algorithm allows correct robots to move in any direction, while keeping an approximation of the polygon. Unlike previous works (e.g., [3, 6]), our algorithm is fault-tolerant, and tolerates permanent crash failures of robots. The only drawback of our algorithm is the fact that it does not permit the rotation of the polygon by the robots, and this is due to the restrictions made on the algorithm in order to ensure the agreement on the ranking by robots. The existence of such algorithm is left as an open question that we will investigate in our future work.

Finally, our work opens new interesting questions, for instance it would be interesting to investigate how to support flocking in a model in which robots may crash and recover.

Acknowledgments

This work is supported by the JSPS (Japan Society for the Promotion of Science) postdoctoral fellowship for foreign researchers (ID No.P 08046).

References

1. Daigle, M. J., Koutsoukos, X. D., Biswas, G.: Distributed diagnosis in formations of mobile robots. IEEE Transactions on Robotics, **23** (2) (2007) 353–369
2. Coble, J., Cook, D.: Fault tolerant coordination of robot teams. Available at: citeseer.ist.psu.edu/coble98fault.html
3. Gervasi, V., and Prencipe, G.: Coordination without communication: the Case of the Flocking Problem. Discrete Applied Mathematics, **143**, (1-3) (2004) 203–223
4. Hayes, A. T., Dormiani-Tabatabaei, P.: Self-organized flocking with agent failure: Off-line optimization and demonstration with real robots. In Proc. IEEE International Conference on Robotics and Automation, **4** (2002) 3900–3905
5. Saber, R. O., Murray, R. M.: Flocking with Obstacle Avoidance: Cooperation with Limited Communication in Mobile Networks. In Proc. 42nd IEEE Conference on Decision and Control, (2003) 2022–2028
6. Canepa, D., Potop-Butucaru, M. G.: Stabilizing flocking via leader election in robot networks. In: Proc. 9th Intl. Symp. on Stabilization, Safety, and Security of Distributed Systems (SSS'07), LNCS 4838 (2007) 52–66
7. Défago, X., Gradinariu, M., Messika, S., Raipin-Parvédy, P.: Fault-tolerant and self-stabilizing mobile robots gathering. In: Proc. 20th Intl. Symp. on Distributed Computing (DISC'06), LNCS 4167 (2006) 46–60
8. Prencipe, G.: CORDA: Distributed Coordination of a Set of Autonomous Mobile Robots. In Proc. European Research Seminar on Advances in Distributed Systems, (2001) 185–190

9. Flocchini, P., Prencipe, G., Santoro, N., Widmayer, P.: Pattern Formation by Autonomous Robots Without Chirality. In Proc. 8th Intl. Colloquium on Structural Information and Communication Complexity (SIROCCO'01) (2001) 147–162
10. Gervasi, V., Prencipe, G.: Flocking by A Set of Autonomous Mobile Robots. Technical Report, "Dipartimento di Informatica, Universitá di Pisa, Italy, (2001) TR-01-24
11. Reynolds, C. W.: Flocks, Herds, and Schools: A Distributed Behavioral Model. Journal of Computer Graphics **21** (1) (1987) 79–98
12. Brogan, D. C., Hodgins, J. K.: Group Behaviors for Systems with Significant Dynamics. Autonomous Robots Journal **4** (1997) 137–153
13. John, T., Yuhai, T.: Flocks, Herds, and Schools: A Quantitative Theory of Flocking. Physical Review Journal, **58** (4) (1998) 4828–4858
14. Yamaguchi, H., Beni, G.: Distributed Autonomous Formation Control of Mobile Robot Groups by Swarm-based Pattern Generation. In Proc. 2nd Int. Symp. on Distributed Autonomous Robotic Systems (DARS'96) (1996) 141–155
15. Dieudonné, Y., Petit, F.: A Scatter of Weak Robots. Technical Report, LARIA, CNRS, France (2007) RR07-10
16. Chandra, T. D., Toueg, S.: Unreliable Failure Detectors for Reliable Distributed Systems. Journal of the ACM, **43** (2) (1996) 225–267
17. Schreiner, K.: NASA's JPL Nanorover Outposts Project Develops Colony of Solar-powered Nanorovers. In IEEE DS Online, **3** (2) (2001)
18. Konolige, K., Ortiz, C., Vincent, R., Agno, A., Eriksen, M., Limketkai, B., Lewis, M., Briesemeister, L., Ruspini, E., Fox, O., Stewart, J. Ko. B., Guibas, L.: CENTIBOTS: Large-Scale Robot Teams. In Journal of Multi-Robot Systems: From Swarms to Intelligent Autonoma, (2003)
19. Bellur, B. R., Lewis, M. G., Templin, F. L.: An Ad-hoc Network for Teams of Autonomous Vehicles. In Proc. 1st IEEE Symp. on Autonomous Intelligent Networks and Systems, (2002)
20. Jennings, J. S., Whelan, G., Evans, W. F.: Cooperative Search and Rescue with a Team of Mobile Robots. In Proc. 8th International Conference on Advanced Robotics, (1997) 193–200

A Perfect Failure Detection

Lemma 11. *If some robot r_i crashes at time t_{crash}, then there is a time t_{mute} after which every correct robot detects the crash of robot r_i, that is: $\exists t_{mute}, t_{mute} \leq t_{crash} + t_{max}$, where t_{max} is the maximum time required for the slowest robot to detect the crash.*

Proof (Lemma 11). Let r_i be a crashed robot. Then, r_i will remain at its current position forever. Let r_f be a correct robot which is the *fastest* robot in the system. By hypothesis on the system model, the time between two consecutive activations of any robot is finite. Then, by definition of the k-bounded scheduler, and Assumption 3, robot r_f detects that r_i has crashed after $(k+1)$ activations (activations of r_f), which takes finite time.

Now, let r_s be a correct robot which is the *slowest* robot in the system. Assume that r_s is activated the least, i.e., r_s is activated only once during the k activations of robot r_f. Then, robot r_s detects that r_i has crashed after $k(k+1)$ activations (activations of r_s), which is also done in finite time. As a result, we can deduce that any correct robot r_j in the system detects the crash of robot r_i in finite time by similar arguments.

Assume that r_i crashes at time t_{crash}, and t_{max} is the maximum time required for $k(k+1)$ activations of the slowest robot, then we can compute t_{mute}, which is

the time after which all correct robot detect the crash of robot r_i as follows: $t_{mute} \leq t_{crash} + t_{max}$. Since after time t_{crash}, robot r_i never moves, then after time t_{mute}, r_i will be permanently suspected by all correct robots in the system. This proves the lemma. □

As a direct consequence from Lemma 11, we derive the following theorem:

Theorem 1 Strong completeness: *eventually every robot that crashes is permanently suspected by every correct robot.*

Algorithm 4 has also the following property:

Theorem 2 Strong accuracy: *there is a finite time after which correct robots are not suspected by any other correct robots.*

Proof (Theorem 2). Let r_i and r_j be two correct robots. Assume without loss of generality that robot r_i is activated only once during k activations of robot r_j. If robot r_i is correct, then by Assumption 3, and by the definition of $k-$bounded scheduler, r_i must move by a non zero distance during the k activations of robot r_j. Also, by Lemma 1, r_i must have finished its move before the start of the $k+1$ activation of r_j. In addition, r_j cannot move to the position that was occupied by r_i by Assumption 3. Since r_j is also correct, it will realize that r_i has changed its position at or before the $k+1$ activation of r_j. Since the time required for the $k+1$ activations of r_j is also finite, r_j will realize that r_i is a correct robot in finite time. □

B Agreed Ranking for Robots

Lemma 2 *Algorithm 2 gives a unique ranking to every correct robot in the system.*

Proof (Lemma 2). The proof is trivial. Since, robots agree on the direction and orientation of the $y-$axis, then, by Algorithm 2, all robots with different $y-$coordinates will have different ranks. In addition, for robots who have the same $y-$coordinate, the clockwise direction is used to determine the sequence. Since, robots agree on the clockwise direction, then two distinct robots having the same $y-$coordinate cannot see each other in the same direction. Thus, a unique ranking is given to each of these robots. □

Lemma 3 *By Algorithm 2, there is a finite time after which, all correct robots agree on the same initial sequence of ranking,* $RankSequence$.

Proof (Lemma 3). Assume without loss of generality that robot r_i is the first robot that was activated by the scheduler. That is, r_i has seen the initial configuration of the robots. Then, the proof consists of showing that all other robots (correct) compute the same sequence of ranking as r_i. We first show that Algorithm 2 preserves the same sequence of ranking computed by r_i. Assume that robot r_j is activated after robot r_i has finished one full cycle. Recall that r_i has changed its position based on Assumption 3. Then, we will show that robot r_j will compute the same rank sequence as r_i no matter what the movement taken by r_i:

1. Robot r_i moves toward $Right(r_i)$: since such a move does not change the $y-$coordinate of r_i, and r_j executes the same algorithm as r_i, then r_j will compute the same rank sequence as r_i.

2. Robot r_i moves toward the closest robot to $Right(r_i)$, say r_c by the distance $min(\epsilon_r/(k+1)(k+2), dist(r_i, r_c)/(k+1)(k+2))$: Assume the worst case where r_i is activated $k+1$ times, while r_c is activated only once. In $k+1$ activations, r_i travels toward r_c by less than the distance $(k+1)dist(r_i, r_c)/(k+1)(k+2)$. Since such a distance is less than $dist(r_i, r_c)$, then the moves performed by r_i during its $k+1$ activations do not change the order of r_i and r_c with respect to left and right, and also it preserves the same $y-$coordinate of r_i and r_c. Thus, by similar arguments as above, r_j will compute the same rank sequence as r_i.

The same proof applies to the other robots that are activated after r_i and r_j, by similar arguments. By Lemma 2, the rank sequence, $RankSequence$ computed by all correct robots is unique. In addition, by the assumption of the k-bounded scheduler, a robot is activated at least once during k activations, and its cycle is finite by Assumption 2. Consequently, after k activations of the same robot in the system, every other correct robot is activated at least once, and have computed the same ranking sequence. Such computation is done in finite time, and the lemma follows. □

Lemma 4 *The ranking algorithm (Algorithm 2) guarantees no collisions between the robots in the system.*

Proof (Lemma 4). The proof is straightforward. The only movement allowed by Algorithm 2 is to make robots move along the perpendicular of their $y-$axes, toward their right. Assume that robot r_i is one of the robots in the system. There are two cases to consider, depending on whether robot r_i has robots on $Right(r_i)$ or not. First, assume that robot r_i has a different y coordinate from the other robots in the system. Then, it is trivial that r_i will not collide with any of these robots because they do not belong to its line of movement. In addition, they will not arrive at its line of movement because they move in parallel to the y-axis of r_i, by Algorithm 2.

Now assume that robot r_i has the same y coordinate as another robot in the system, say r_j. Assume without loss of generality that r_j is the closest to r_i in $Right(r_i)$. By Algorithm 2, r_i is allowed to move at each activation cycle by at most $min(dist(r_i, r_j)/(k+1)(k+2), \epsilon_r/(k+1)(k+2))$. Then, even in the worst case when r_i is activated each time during k activations in the system, while r_j is activated the least, we will not have the situation where r_i collides with r_j after k activations of r_i because r_i will not reach r_j since the distance $kdist(r_i, r_j)/(k+1)(k+2)$ is less than $dist(r_i, r_j)$. □