Fault-Tolerant Gathering Algorithms for Autonomous Mobile Robots

(Extended Abstract)

Noa Agmon [*] David Peleg [*] [†]

Abstract

This paper studies fault tolerant algorithms for the problem of gathering N autonomous mobile robots. A gathering algorithm, executed independently by each robot, must ensure that all robots are gathered at one point within finite time. It is first observed that most existing algorithms fail to operate correctly in a setting allowing crash failures. Subsequently, an algorithm tolerant against one crash-faulty robot in a system of three or more robots is presented. It is then shown that in an asynchronous environment it is impossible to perform a successful gathering in a 3-robot system with one Byzantine failure. Finally, in a fully synchronous system, an algorithm is provided for gathering $N \geq 3$ robots with at most a single faulty robot, and a more general gathering algorithm is given in an N-robot system with up to f faults, where $N \geq 3f + 1$.

1 Introduction

1.1 Background Systems of multiple autonomous mobile robots engaged in cooperative activities have been extensively studied throughout the past decade [2, 8, 4, 5, 1]. This subject is of interest for a number of reasons. For one, it may be possible to use a multiple robot system in order to accomplish tasks that no single spatially limited robot can achieve. Another advantage of multiple robot systems has to do with the decreased cost due to the use of simpler and cheaper individual robots. Also, these systems have immediate applicability in a wide variety of tasks, such as military operations and space missions. Subsequently, studies of autonomous mobile robot systems can be found in different disciplines, from engineering to artificial intelligence (e.g., [9]).

Our interest is in problems related to the *distributed control* of systems of autonomous mobile robots. Most studies on robot control problems resulted in the design of algorithms based on heuristics, with little emphasis on formal analysis of the correctness, termination or complexity properties of the algorithms. During the last few years, various aspects of this problem have been studied from the point of view of distributed computing (cf. [2, 10, 15, 16, 13]), where the focus is on trying to model an environment consisting of mobile robots, and studying the capabilities the robots must have in order to achieve their common goal. A number of computational models were proposed in the literature, and some studies attempted to characterize the influence of the models on the ability of a group of robots to perform certain basic tasks under different constraints.

The primary motivation of the studies presented in [13, 16, 10, 11, 15] is to identify the minimal capabilities a collection of distributed robots must have in order to accomplish certain basic tasks and produce interesting interaction. Consequently, the models adopted in these studies assume the robots to be relatively weak and simple. In particular, these robots are generally assumed to be dimensionless (namely, treated as points that do not obstruct each other's visibility or movement), oblivious (or memoryless, namely, do not remember their previous actions or the previous positions of the other robots), have no common coordinate system, orientation or scale, no explicit communication, and are anonymous (some of these assumptions are modified in order to achieve goals that are otherwise unfeasible). Thus the robots base their movement decisions on viewing their surroundings and analyzing the configuration of robot locations. A robot is capable of locating all robots within its visibility range and laying them in their private coordinate system, thereby calculating their position (distance and angles) with respect to one another and with respect to itself. Hence, from the "distributed computing" angle, such problems are particularly interesting since they give rise to a different type of communication model, based solely on "positional" or "ge-

[*]Department of Computer Science and Applied Mathematics, The Weizmann Institute of Science, Rehovot, 76100 Israel. E-mail: {noas,peleg}@wisdom.weizmann.ac.il.

[†]Supported in part by a grant from the Israel Science Foundation.

ometric" information exchange.

A basic task that has received considerable attention is the *gathering* problem, defined as follows. Given an initial configuration of N autonomous mobile robot, all N robots should occupy a single point within a finite number of steps. The closely related *convergence* problem is defined similarly, except that the robots are only required to converge to a single point, rather than reach it. Namely, instead of demanding that the robots gather to one point within finite time, the convergence requirement is that for every $\epsilon > 0$, there is a time t_ϵ by which all robots are within distance of at most ϵ of each other.

1.2 Fault-tolerance As the common models of multiple robot systems assume cheap, simple and relatively weak robots, the issue of resilience to failure becomes prominent, since in such systems one cannot possibly rely on assuming fail-proof hardware or software, especially when such robot systems are expected to operate in hazardous or harsh environments. At the same time, one of the main attractive features of multiple-robot systems is their potential for enhanced fault tolerance. It seems plausible that the inherent redundancy of such systems may be exploited in order to enable them to perform their tasks even in the presence of faults.

Following the common "f of N" classification often used in the area, a fault tolerant algorithm for a given task is required to ensure that in a system consisting of N robots where it is assumed that at most f robots might fail in any execution, the task is achieved by all nonfaulty robots, regardless of the actions taken by the faulty ones. In the gathering task, for example, when faults are introduced into the system, the requirement applies only for the nonfaulty robots, i.e., if f' robots fail, then all the remaining $N - f'$ nonfaulty robots are required to occupy a single point within a finite time.

Perhaps surprisingly, however, this aspect of multiple robot systems has been explored to very little extent so far. In fact, almost all the results we are aware of in the literature rely on the assumption that all robots function properly, and follow their protocol without any deviation. One exception concerns *transient* failures. As observed in [16, 13, 6], any algorithm that works correctly on oblivious robots is necessarily *self-stabilizing*, i.e., it guarantees that after any transient failure the system will return to a correct state and the goal will be achieved. Yet another line of study concerns a fault model where it is assumed that restricted sensor and control failures might occur, but if faults do occur in the system, then the identity of the faulty robots becomes known to all robots [13]. This may be an unrealistic assumption in many typical settings, and it clearly provides an easy means of overcoming the faults: each nonfaulty robot may simply ignore the failed ones, effectively removing them from the group of robots, so the algorithm continues to function properly. However, in case unidentified faults occur in the system, it is no longer guaranteed that the algorithms of [13, 16] remain correct, i.e., the goal might not be achieved. The only concrete attempt we're aware of for dealing with crash faults is described in [18], where an algorithm is given for the *Active Robot Selection Problem (ARSP)* in the presence of initial crash faults. The ARSP creates a subgroup of nonfaulty robots from a group that includes also initially crashed robots and makes the robots in that subgroup recognize one another. This allows the nonfaulty robots in the subgroup to overcome the existence of faults in the system, and they can further execute any algorithm within the group.

Hence the design of fault-tolerant distributed control algorithms for multiple robot systems is still a largely unexplored direction, which the current paper aims at investigating.

1.3 Related work The class of agreement and pattern formation problems of autonomous mobile robot systems discussed so far in the literature includes, among others, the following basic tasks: formation of simple geometric patterns [13], gathering and convergence [14, 15, 4, 3], flocking (namely, following the movement of a predefined leader) [12], uniform distribution over a specified region [13], and partitioning into groups [13].

The problem of gathering autonomous mobile robots, dealt with in this article, was studied extensively in two computational models. The first is the model of [13, 16], hereafter referred to as the *semi-synchronous* ($\mathcal{SSYNC}$) model. The second is the closely related CORDA model [10, 11, 15], hereafter referred to as the *asynchronous* ($\mathcal{ASYNC}$) model.

The gathering problem was first discussed in [15, 16] in the $\mathcal{SSYNC}$ model. It was proven there that it is impossible to achieve gathering of *two* oblivious autonomous mobile robots that have no common sense of orientation in the $\mathcal{SSYNC}$ model. The algorithms presented therein for $N \geq 3$ robots rely on the assumption that a robot can identify a point p^* occupied by two or more robots (a.k.a. *multiplicity point*). This assumption was later proven to be essential for achieving gathering in all asynchronous and semi-synchronous models [12]. Another necessary requirement for solvability in the $\mathcal{SSYNC}$ and $\mathcal{ASYNC}$ models is that the input configuration does not include more than one multiplicity point of nonfaulty robots (it is easy to show that if two multiplicity points of nonfaulty robots are allowed, the situation is equivalent to the 2-robot system, thus gathering is impossible). In fact, all known gathering algorithms for $N \geq 3$ rely on a strategy by which a single multiplicity point p^* is formed during the execution of the algorithm, and once this happens, all robots move to the point p^*.

Under these assumptions, an algorithm is developed in [16] for gathering $N \geq 3$ robots in the $\mathcal{SSYNC}$ model. In the $\mathcal{ASYNC}$ model, an algorithm for gathering $N = 3, 4$ robots is brought in [12, 4], and an algorithm for gathering $N \geq 5$ robots has recently been described in [3].

1.4 Our results This paper presents a systematic study of failure-prone robot systems, through examining the gathering problem in the crash and Byzantine fault models. An (N, f)-*fault system* is a system consisting of N robots, at most f of which might fail at any execution. An (N, f)-*crash system* (resp., (N, f)-*Byzantine system*), is an (N, f)-fault system where the faults considered are according to the crash or Byzantine model.

In the crash fault model, we show that the gathering problem is solvable in current computational models such as the $\mathcal{SSYNC}$ model, though most existing algorithms fail to deal correctly with such faults and, in particular, there is currently no algorithm for $N \geq 4$ that solves the gathering problem in the presence of one faulty robot under the crash fault model. We propose an algorithm that solves the gathering problem in $(N, 1)$-crash system, for any $N \geq 3$, under the $\mathcal{SSYNC}$ model.

We then consider (N, f)-Byzantine systems for $N \geq 3$. We first observe that all existing algorithms fail to deal correctly with this situation. Moreover, we show that it is impossible to perform a successful gathering in $(3, 1)$-Byzantine systems under the $\mathcal{SSYNC}$ model. We then introduce the *fully synchronous* ($\mathcal{FSYNC}$) model, which is similar to the synchronous model mentioned in [16], and present an algorithm solving the gathering problem under this model in (N, f)-Byzantine systems for every [1] $N \geq 3f + 1$.

2 The model

Each robot R_i in the system is assumed to operate individually in simple cycles. Every cycle consists of three steps, "look", "compute" and "move". In the $\mathcal{FSYNC}$ and $\mathcal{SSYNC}$ models the length of this cycle is uniform for all robots. The "look" step identifies the locations of all robots in R_i's private coordinate system; the result of this step is a multiset of points $P = \{p_1, \dots, p_N\}$ defining the current *configuration*. The robots are indistinguishable, so each robot R_i knows its own location p_i, but does not know the identity of the robots at each of the other points. The "compute" step executes the given algorithm, resulting in a goal point p_G. In the "move" step, the robot moves towards the point p_G. The robot might stop before reaching its goal point p_G, but is promised to traverse a distance of at least S (unless it has reached the goal). Note that the "look" and "move" steps are carried out identically in every cycle, and the differences between different algorithms occur in the "compute" step. Moreover, the procedure carried out in the "compute" step is identical for all robots. If the robots are oblivious, then the algorithm cannot rely on information from previous cycles, thus the procedure can be fully specified by describing a single "compute" step, and its only input is the current configuration $P = \{p_1, \dots, p_N\}$, giving the locations of the robots $R_1, \dots, R_N$. Throughout, we denote the location of R_i in the configuration P by $p(R_i)$. Also, whenever no confusion may arise, we identify $p(R_i)$ as the point p_i. As mentioned earlier, our computational model for studying and analyzing problems of coordinating and controlling a set of autonomous mobile robots follows two well studied models: the $\mathcal{SSYNC}$ model and the $\mathcal{ASYNC}$ model. The semi-synchronous ($\mathcal{SSYNC}$) model is partially synchronous, in the sense that all robots operate according to the same clock cycles, but not all robots are necessarily active in all cycles. The activation of the different robots can be thought of as managed by a hypothetical "scheduler", whose only "fairness" obligation is that each robot must be activated and given a chance to operate infinitely often in any infinite execution. The fully asynchronous ($\mathcal{ASYNC}$) model differs from the $\mathcal{SSYNC}$ model in that each robot acts independently in a cycle composed of *four* steps: Wait, Look, Compute, Move. The length of this cycle is finite, but not bounded. Consequently, there is no bound on the length of the walk in a single cycle, and different cycles of the same robot may vary in length. In contrast, in the $\mathcal{SSYNC}$ model a bound exists on the cycle length due to the common clock and as a result the robot will not necessarily reach the target point p in the current cycle, but stop somewhere on its trajectory to p.

In this paper we consider also the extreme *fully synchronous* ($\mathcal{FSYNC}$) model. This model is similar to the $\mathcal{SSYNC}$ model in the sense that each robot works according to the same three steps cycle. In addition, in this model, robots operate according to the same clock

[1] A peculiarity of our algorithms is that $N = 3$ robots can tolerate $f = 1$ failures, but $N > 3$ robot systems require $N \geq 3f + 1$, rather than $N \geq 3f$.

cycles, and all robots are active on all cycles. There is a lower bound of S units on the minimum movement of a robot in a cycle, except in the case that the robot has reached the point towards which it was moving. There is also a limit of maximum movement in each cycle.

Following most previous papers on the gathering problem in the literature [14, 15, 7, 4, 12], the model adopted throughout this paper is the *oblivious* model, where it is assumed that the robots cannot remember their previous states, and thus the decisions they make in each step are based only on the current configuration. The main motivation for developing algorithms for the oblivious model is twofold. First, solutions developed on the basis of assuming non-obliviousness do not necessarily work in a dynamic environment where the robots are activated in different cycles, or robots might be added/removed from the system dynamically. Secondly, as mentioned earlier, any algorithm that works correctly for oblivious robots is inherently self-stabilizing, i.e., it withstands transient errors. More generally, it is advantageous to develop algorithms for the weakest robot types possible, as an algorithm that works correctly for weak robots will clearly work correctly in a system of stronger robot types. In contrast, our lower bounds serve mainly to draw the borderlines where the various models become too weak to allow solutions.

The fault models discussed throughout the paper are the *crash* fault model and the *Byzantine* fault model. In the *Byzantine* fault model, it is assumed that a faulty robot might behave in arbitrary and unforeseeable ways. For the sake of analysis, it is convenient to model the behavior of the system by means of an *adversary* which has the ability to control the behavior of the faulty robots, as well as the "undetermined" features in the behavior of the nonfaulty processors (e.g., the distance to which they move). Specifically, in each cycle the adversary has the following roles. For each nonfaulty robot, it determines the distance to which the robot will move in this cycle (for a robot R_i located at p_i and headed for the goal point p_G, if $\text{dist}(p_i, p_G) \leq S$, then the robot must be allowed to reach p_G; else, the adversary may stop R_i at any point on the line segment $\overline{p_i p_G}$ that is at least in distance S away from p_i). In addition, for each faulty robot it determines its course of action in that cycle, which can be arbitrary. In the *crash* fault model, the behavior of the system is similar to the one described in the Byzantine fault model, except that for each faulty robot the adversary is only allowed to stop its movement. This may be done at any point in time during the cycle, i.e., either during the movement towards the goal point or before it has started. Once the adversary crashed the faulty robot, that robot will remain stationary indefinitely.

3 Gathering in the crash fault model

3.1 Inadequacy of known algorithms Let $\mathcal{A}$ be a gathering algorithm that works in an N-robot system. In every configuration C, the algorithm instructs some robots to move and some to remain stationary. Denote the number of robots $\mathcal{A}$ instructs to move in configuration C by $M(C, \mathcal{A})$, and let $\check{M}(N, \mathcal{A}) = \min\{M(C, \mathcal{A}) \mid C \text{ is a configuration in an } N\text{−robot system}\}$.

LEMMA 3.1. *In an (N, f)-crash system, an algorithm $\mathcal{A}$ with $\check{M}(N, \mathcal{A}) \leq f$ will fail in achieving gathering or convergence.*

(Throughout this extended abstract, proofs are deferred to the full paper.)

We remark that every gathering algorithm $\mathcal{A}$ we're aware of in the $\mathcal{SSYNC}$ and $\mathcal{ASYNC}$ models [14, 15, 4, 3] has $\check{M}(N, \mathcal{A}) = 1$ for $N \geq 4$, and consequently, by Lemma 3.1, these algorithms fail to achieve gathering even in the presence of one crash faulty robot. On the other hand, it turns out that the gathering algorithm given in [4] for $N = 3$ in the $\mathcal{ASYNC}$ model, can be shown to operate correctly also in the presence of one crashed robot (we give a slightly simpler algorithm for this case in the $\mathcal{SSYNC}$ model below), and the algorithm given therein for $N = 4$ can be transformed into an algorithm for $(4, 1)$-crash systems with some minor changes. The algorithm described in [3] for gathering $N \geq 5$ robots in the $\mathcal{ASYNC}$ model uses the operation moveToIfFreeWay which can fail even in the presence of one crash faulty robot, if that robot lies between any robot R_i and its goal point p.

An additional difficulty in handling a robot system with crash faults is caused by the assumption, made by all current algorithms, that only a single multiplicity point is created throughout the execution of the algorithm. In the presence of faults, the fact that the adversary has the ability to stop the nonfaulty robots after traversing a minimal distance S and the ability to crash the faulty robots at any step during the execution, makes it easy for the adversary to create a second multiplicity point once the first was created, whenever the trajectories of two or more robots moving towards their goals intersect.

3.2 An algorithm for a $(3, 1)$-crash system Consider the following Procedure 3-Gather$_{\text{crash}}$ for gathering in a $(3, 1)$-crash system in the $\mathcal{SSYNC}$ model. As discussed earlier, we only need to present the procedure used for the "compute" step. The input to this procedure is the configuration $P = \{p_1, p_2, p_3\}$. The procedure classifies the configuration according to its state, and acts in each case as follows.

Procedure 3-Gather$_{\mathbf{crash}}(P)$

1. **State [Mult]:** P contains a multiplicity point p^*: Set $p_G \leftarrow p^*$.
2. **State [Collinear]:** p_1, p_2, p_3 are collinear, say, with p_2 in the middle: Set $p_G \leftarrow p_2$.
3. **State [Obtuse]:** $\exists i \in \{1, 2, 3\}$ such that $\angle p_j p_i p_k \geq \pi/2$: Set $p_G \leftarrow p_i$.
4. **State [Acute]:** $\triangle p_1, p_2, p_3$ is acute: Set p_G to be the intersection point of the three angle bisectors.

Remark: Note that state [Collinear] is redundant, since it is covered by state [Obtuse]. It is included merely for convenience of presentation.

In analyzing our algorithms, we use the following notation regarding points and lines in the Euclidean plane. Denote the Euclidean distance between two points p and q by $\text{dist}(p, q)$. Also, denote the Euclidean distance between two current locations p_i and p_j of the two robots R_i and R_j (respectively) by $\text{dist}(R_i, R_j)$. Denote the line segment between the points p and q by $\overline{pq}$.

Our analysis is based on showing that in each case, some pair of robots decrease their distance by a constant (depending on the minimum movement length value S) while no other pair of robots increase their distance, so eventually they must meet. A fact used throughout the analysis is that the algorithm guarantees that the states the system can be in create a directed acyclic graph (DAG) leading to the state [Mult], and the goal point of each robot remains constant until a multiplicity point is created. (This property does not hold, for example, in case the goal in each cycle is chosen to be the center of gravity or the intersection point of the three angle bisectors).

THEOREM 3.1. *Algorithm* 3-Gather$_{crash}$ *solves the gathering problem in a* $(3, 1)$*-crash system under the* $\mathcal{SSYNC}$ *model.*

3.3 An algorithm for an $(N, 1)$-crash system, $N \geq 3$

Let us start with some terminology. A *legal* configuration in the $(N, 1)$-crash system is a set P of robot locations that has at most one multiplicity point. Denote the *smallest enclosing circle* of the set P by $\mathcal{SEC}(P)$ and the points on its circumference by $\mathcal{C}_{cir}(P)$ and let $\mathcal{C}_{int}(P) = P \setminus \mathcal{C}_{cir}(P)$. For a circle C and the set of points $P = \{p_1, \ldots, p_l\}$ on its circumference, denote the partition of the circle C into Voronoi cells according to the points in P by $\text{Vor}(C, P)$, and denote by $\text{Cell}(p_k)$ the cell defined by the point $p_k \in P$ (see Figure 1). Two points q and q' in C are said to share the cell $\text{Cell}(p_k)$ if they both lie inside the cell or on its boundary.

Consider the following Algorithm Gather$_{\text{crash}}$ for gathering all nonfaulty robots in an $(N, 1)$-crash system

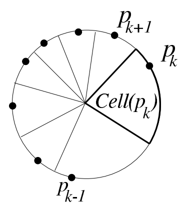

Figure 1: Example of a circle division according to the Voronoi cells.

under the $\mathcal{SSYNC}$ model. The input to this algorithm is a legal configuration $P = \{p_1, \ldots, p_N\}$. The algorithm classifies the configuration according to its state, and acts in each case as follows.

Algorithm Gather$_{\mathbf{crash}}(P)$

1. **State [Singletons]:** P does not contain a multiplicity point: Invoke Procedure Create_Mult(P).
2. **State [MULT]:** P contains a single multiplicity point p^*: Invoke Procedure GoTo_Mult(P).

Procedure Create_Mult(P)

State [N3]: $N = 3$: Invoke Procedure 3-Gather$_{\text{crash}}$ on p_1, p_2, p_3.

State [N4+]: $N \geq 4$:

1. Compute the smallest enclosing circle of the set P, $\mathcal{SEC}(P)$.
2. **Sub-State [IN0]:** $|\mathcal{C}_{int}(P)| = 0$: Set p_G to be the center of $\mathcal{SEC}(P)$.
3. **Sub-State [IN1]:** $|\mathcal{C}_{int}(P)| = 1$ with p_j as the single point in $\mathcal{C}_{int}(P)$: Set $p_G \leftarrow p_j$
4. **Sub-State [IN2]:** $|\mathcal{C}_{int}(P)| = 2$ with p_i and p_j as the two points in $\mathcal{C}_{int}(P)$:
   Each robot R_k in $\mathcal{C}_{cir}(P)$ sets $p_G(R_k) \leftarrow p(R_k)$.
   The two robots R_i and R_j in $\mathcal{C}_{int}(P)$ do:
   (a) Compute the Voronoi partition $\text{Vor}(\mathcal{SEC}(P), \mathcal{C}_{cir}(P))$.
   (b) **Sub-State [IN2(a)]:** p_i and p_j do not share cells: R_i and R_j move towards the center of $\mathcal{SEC}(P)$.
   (c) **Sub-State [IN2(b)]:** p_i and p_j share a single cell, $\text{Cell}(R_k)$: R_i and R_j move towards R_k.
   (d) **Sub-State [IN2(c)]:** p_i and p_j share two cells, i.e., both robots lie on the radius forming the boundary between two adjacent cells $\text{Cell}(R_k)$ and $\text{Cell}(R_{k+1})$:
   The robot closer to the circle, say R_i, chooses the first of R_k, R_{k+1} in its clockwise direction, say R_k, and sets $p_G(R_i) \leftarrow p(R_k)$; The other robot, R_j, sets $p_G(R_j) \leftarrow p(R_i)$.
5. **Sub-State [IN3]:** $|\mathcal{C}_{int}(P)| \geq 3$:
   Each robot R_k in $\mathcal{C}_{cir}(P)$ sets $p_G(R_k) \leftarrow p(R_k)$.
   Each robot R_k in $\mathcal{C}_{int}(P)$ recursively invokes Procedure Create_Mult($\mathcal{C}_{int}(P)$).

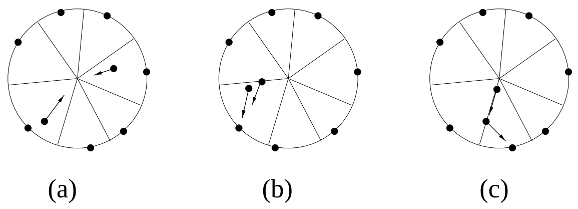
Figure 2: Illustration of the three sub-states of sub-state [**IN2**] in Procedure Create_Mult.

We say that robot R_i, has a *"free corridor"* to the point p if no other robot is currently located on the straight line segment $\overline{p_i p}$. Note that as robots are viewed as dimensionless objects, the availability of a free corridor is not necessarily a prerequisite for allowing a robot to get home free. However, allowing a robot to follow a trajectory through the location of another robot makes the algorithm prone to the creation of more than one multiplicity point. Therefore Procedure GoTo_Mult attempts to avoid such trajectories.

Procedure GoTo_Mult(P) **(for robot R_i)**

```
/* The configuration contains a multiplicity point $p^*$ */
1. State [Free]: $R_i$ has a free corridor to $p^*$: Set $p_G \leftarrow p^*$.
2. State [Blocked]: There exist one or more robots on $R_i$'s trajectory towards $p^*$:
    a. Translate your coordinate system to be centered at $p^*$.
    b. Compute for each robot $R_j$ the angle $\mu_j$ of $\overrightarrow{p^* p_j}$ counterclockwise from the $x$ axis.
    c. Find the robot $R_k$ with smallest angle $\mu_k > \mu_i$. Let $\mu = (\mu_k + 2\mu_i)/3$, and $d = \text{dist}(R_i, R_k)$ (see Figure 3(a)).
    d. Let $p'_i$ be the point at distance $d$ and angle $\mu$ from $p^*$.
    e. Set $p_G(R_i) \leftarrow p'_i$.
```

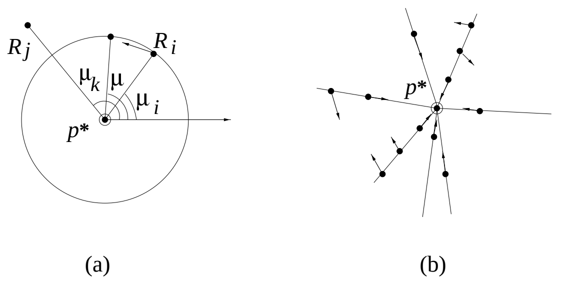

Figure 3: Illustration of state [**Blocked**] in Procedure GoTo_Mult.

To analyze the algorithm, we show that in an $(N,1)$-crash system, if the initial configuration contains no multiplicity points, then Procedure Create_Mult leads, within finite time, to a configuration that includes a single multiplicity point p^*, and subsequently, within finite time, procedure GoTo_Mult gathers all nonfaulty robots at p^* while avoiding the creation of additional multiplicity points.

THEOREM 3.2. *Algorithm* Gather$_{crash}$ *solves the gathering problem in an* $(N,1)$*-crash system under the* $\mathcal{SSYNC}$ *model for any* $N \geq 3$.

4 Impossibility of gathering under the Byzantine fault model

In [11] it is shown that the class of problems solvable in $\mathcal{ASYNC}$ is contained in the class of problems solvable in the $\mathcal{SSYNC}$ model. We next prove that in the $\mathcal{SSYNC}$ model it is impossible for any algorithm to achieve either gathering or convergence of three robots in the Byzantine fault model, even in the presence of at most one faulty robot.

Definition: A gathering algorithm $\mathcal{A}$ is called *hyperactive* if it instructs every robot to make a move in every cycle until the task is achieved, i.e., $\check{M}(N,\mathcal{A}) = N$.

THEOREM 4.1. *In a* $(3,1)$*-Byzantine system under the* $\mathcal{SSYNC}$ *model, any non-hyperactive gathering algorithm will fail in achieving gathering or convergence.*

Definition: A distributed robot algorithm is N-*diverging* if there exists an (N,f)-Byzantine system and a configuration in which the instructions of the algorithm combined with the actions of the adversary can cause two nonfaulty robots to increase the distance between them. (An example for divergence caused by the instructions of the algorithm is illustrated in Figure 4(a). An example for divergence caused by the intervention of the adversary is illustrated in Figure 4(b), where robot R_1 is stopped short of reaching its goal point.)

The premature-stopping technique: Our impossibility proofs make extensive use of the following technique. In order to cause two robots to diverge in some given configuration C of a given system $\mathcal{T}$, the adversary can stop a nonfaulty robot R_i after traversing a relatively small distance S_i. Note that S_i might be smaller than S in the current system, in which case the adversary is not permitted to stop R_i prematurely. However, as the algorithm is required to be valid in any system, it is intuitively clear that we may always consider a different system $\mathcal{T}'$ with S small enough to allow a movement of distance S_i. Moreover, since the algorithm is unaware of the value of S, it cannot distinguish between identical configurations in the two systems $\mathcal{T}$ and $\mathcal{T}'$, and will issue the same instructions to each robot in configuration C in $\mathcal{T}$ and $\mathcal{T}'$. Therefore the premature-stopping technique can be applied with any movement length.

As for the applicability of the premature-stopping technique, the adversary can apply it to cause the robots to diverge in any case where two robots move towards their respective goals on nonintersecting trajectories (see Figure 4(c)). In addition, even if the trajectories do intersect, the adversary can still apply the technique in some cases and again cause divergence, as seen in Figure 4(b).

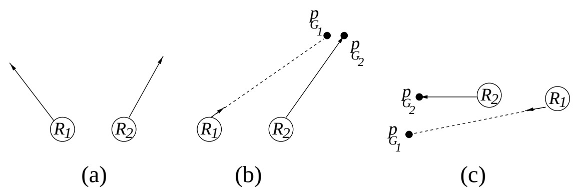

Figure 4: Divergence of robots.

LEMMA 4.1. *In the $\mathcal{SSYNC}$ (or even the $\mathcal{FSYNC}$) model, a 3-diverging algorithm will fail to achieve gathering or convergence.*

THEOREM 4.2. *In a $(3,1)$-Byzantine system under the $\mathcal{SSYNC}$ model it is impossible to perform successful gathering or convergence.*

Remarks: In the $\mathcal{FSYNC}$ model, an N-diverging algorithm for $N > 3$ will not necessarily fail. In particular, the algorithms suggested in Sect. 6 for the $\mathcal{FSYNC}$ model might be diverging, yet still achieve gathering. Also, in the $\mathcal{FSYNC}$ model, a non-hyperactive algorithm will not necessarily fail. In particular, the gathering algorithm for $N = 3$ suggested in Sect. 5 for the $\mathcal{FSYNC}$ model is not always hyperactive (for example, when the 3 robots are collinear, the robot lying in the middle is instructed to remail still). In fact, the converse may hold, namely, in both the $\mathcal{FSYNC}$ and $\mathcal{SSYNC}$ models, a *hyperactive* algorithm might be problematic. For example, it can be shown that in a one-dimensional setting, the adversary can cause failure of every hyperactive algorithm (Proof of this fact is deferred to the full paper.)

Discussion: The difficulty of handling a system of autonomous mobile robots with Byzantine faults lies, among other reasons, in the conventions regarding multiplicities which most existing algorithms rely on. In particular, these algorithms are based on enforcing the following conventions. (a) no more than one multiplicity point is created throughout the execution of the algorithm, until successful gathering is achieved, (b) all robots lying in a multiplicity point remain stationary, and (c) robots lying in a multiplicity point are never separated again. All the above assumptions no longer hold in a system where Byzantine faults might occur. First, the adversary could create a second multiplicity point as soon as it detects one such point, by "failing" a robot that does not lie in the multiplicity point and sending it to the location of yet another currently single robot. As a result, in the gathering problem assumption (b) cannot be relied on. Assumption (c) is violated even if the algorithm instructs all robots lying on the same point to move towards the *same* destination point, as the adversary could stop their movement in different locations.

Since all known algorithms rely on conventions (a)-(c) listed above, which can be violated in a system consisting of N robots with at most one Byzantine faulty robot, it is clear that those algorithms fail to achieve gathering.

To get a feeling for the possible complications that may occur in this model, let us consider some simple solutions one might propose for the problem. One natural general approach for attacking the problem is to try to gradually reduce the number of distinct points where the robots reside, by gathering partial subsets of robots at different points. A possible algorithm attempting to achieve that is to require each robot, in each cycle, to move towards its *closest* neighbor. This may lead to deadlocks once the robots pair up, since their closest neighbor already resides at the same location. Therefore the algorithm should instruct each robot to move towards the closest robot among those currently residing at locations *other* than its own. One problem that arises is that sets of robots that have already met might break up again, hence "progress" is hard to measure. Another obvious problem is that of symmetry-breaking. Even ignoring this problem, this approach can still lead to non-converging scenarios. For instance, suppose that the N robots are located on a straight line, with R_i at location $x_i = i(i-1)/2$. Then the algorithm requires R_1 to move towards R_2 and R_i to move towards R_{i-1} for every $2 \leq i \leq N$. However, if R_1 is faulty, and chooses to move *away* from R_2, and all robots traverse exactly distance S, then the configuration is translated by S in the $-x$ direction, and is otherwise unchanged. (See Fig. 5.)

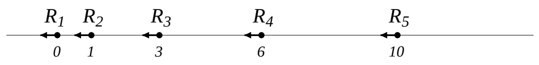

Figure 5: Bad scenario for nearest-neighbor algorithm.

Another natural algorithm is based on computing the center of gravity p_G of the configuration, and going to p_G. This algorithm can be failed by the adversary in similar manner, by failing a robot located far from the rest, and taking it to a walk towards infinity, forcing the entire pack of nonfaulty robots to drag along.

5 Fault tolerant gathering in the $\mathcal{FSYNC}$ model

We now discuss the problem of gathering N autonomous mobile robots in an (N, f)-Byzantine system under the fully synchronous model. We use the following notation. Denote the *geometric span* (or *diameter*) of the set of points P by $\text{Span}(P) = \max\{\text{dist}(p,q) \mid p,q \in P\}$. Denote the convex hull of a multiset of points P by $H(P)$, and the set of vertices of $H(P)$ by $V_H(P)$. Denote the *center of gravity* (or barycenter) of a multiset P of $n \geq 3$ points $p_i = (x_i, y_i), i = 1, \ldots, N$ by $C_{\text{grav}}(P) = ((\sum_{i=i}^{N} x_i)/N \ , \ (\sum_{i=i}^{N} y_i)/N)$.

Note that for any set of points P, while the center of gravity $C_{\text{grav}}(P)$ is defined in terms of the point coordinates in some specific coordinate system, the resulting point is independent of the particular coordinate system in use. Hence for a set of robots in some arbitrary configuration C in the plane, whenever each of the robots computes $C_{\text{grav}}(P)$, the resulting point computed by the different robots is the same, even if each robot has its own coordinate system.

Definition: A distributed robot algorithm is *concentrating* if it satisfies the following properties: (1) it is non-diverging, i.e., no two nonfaulty robots will increase the distance between them in any round, and (2) there exists a constant $c > 0$ such that at each step, at least one pair of nonfaulty robots that are at different locations either meets or decreases the distance between them by at least c.

LEMMA 5.1. *Let $\mathcal{A}$ be a concentrating algorithm. Then in a $(3,1)$-Byzantine system under the $\mathcal{FSYNC}$ model, $\mathcal{A}$ achieves gathering.*

Definition: A distributed robot algorithm $\mathcal{A}$ is said to *dictate triple convergence* in a given cycle if in that cycle it instructs three robots to decrease the distance between every pair of them by a constant amount.

Note that for $N = 3$, triple convergence implies also non-divergence and hence also concentration. Note also that this condition does not require that the robots involved be nonfaulty; in particular, one of them may fail and disobey the algorithm's instructions, in which case its distances will not decrease as needed. Nevertheless, the condition turns out to be sufficient for gathering in certain settings.

Let us next describe a gathering algorithm for three robots in a $\mathcal{FSYNC}$ model with at most one Byzantine fault. The input to this procedure is a configuration $P = \{p_1, p_2, p_3\}$.

Procedure 3-Gather$_{\mathbf{Byz}}(P)$

1. **State [Collinear]:** p_1, p_2, p_3 are collinear with p_2 in the middle: Set $p_G \leftarrow p_2$.
2. **State [Triangle]:** The points form a triangle: Set p_G to be the intersection of its angle bisectors.

Proving this algorithm enjoys triple-convergence yields the following.

THEOREM 5.1. *Algorithm* 3-Gather$_{Byz}$ *solves the gathering problem in a $(3,1)$-Byzantine system under the $\mathcal{FSYNC}$ model.*

6 Gathering algorithm for $f \geq 1$ and $N \geq 3f+1$ in the $\mathcal{FSYNC}$ model

In this section we propose an algorithm for solving the gathering problem in an (N, f)-Byzantine system, where $N \geq 3f+1$ in the $\mathcal{FSYNC}$ model. The main idea of the algorithm is to ensure that the goal point selected in each cycle falls in the convex hull of the *nonfaulty* robot locations. As shown later on, this ensures that the geometric span of the set of locations of the nonfaulty robots decreases by at least $0.25S$, thus the robots will meet within a finite number of cycles. Due to its high complexity, this algorithm is only of theoretical merit, except for small values of f.

Definition: The *hull intersection* $H^k_{\text{int}}(P)$ is the convex set created as the intersection of all $\binom{N}{k}$ sets $H(P \setminus \{p_{i_1}, \ldots, p_{i_k}\})$, for $1 \leq k \leq N$, $p_{i_j} \in P$. (See Figure 6 for $k = 1$.)

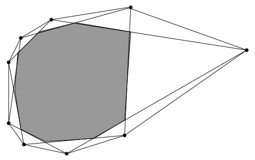

Figure 6: Illustration of H^1_{int}

Consider the following Procedure Gather$_{\mathbf{Byz}}$ for determining the goal point p_G in each cycle. The input to this procedure is a configuration $P = \{p_1, \ldots, p_N\}$, and f is the maximum number of faulty robots.

Procedure Gather$_{\mathbf{Byz}}(P)$

1. Compute $Q \leftarrow V_H(H^f_{\text{int}}(P))$.
2. Set $p_G \leftarrow C_{\text{grav}}(Q)$.

To establish that the algorithm is well-defined, we rely on Helly's theorem (cf. [17]) to prove the following.

LEMMA 6.1. *For a multiset $P = \{p_1, \ldots, p_N\}, N \geq 3k+1$, $H^k_{int}(P)$ is convex and nonempty.*

The analysis of Procedure $\mathsf{Gather}_{\mathrm{Byz}}$ is based on showing that if a set of K robots $R_1, \ldots, R_K$ initially located at the points $P = \{p_1, \ldots, p_K\}$ move towards a point p_G in their convex hull $H(P)$, and their new positions are at the points $P' = \{p'_1, \ldots, p'_K\}$, then their geometric span decreases by at least cS for some constant $c \geq 1/4$, i.e., $\mathrm{Span}(P') \leq \mathrm{Span}(P) - cS$. Consequently, the robots will meet within a finite number of cycles.

THEOREM 6.1. *Algorithm Gather_{Byz} solves the gathering problem in an (N, f)-Byzantine system under the $\mathcal{FSYNC}$ model for any $N \geq 3f + 1$.*

References

[1] E. Arkin, M. Bender, S. Fekete, J. Mitchell, and M. Skutella. The freeze-tag problem: How to wake up a swarm of robots. In *Proc. 13th ACM-SIAM Symp. on Discrete Algorithms*, January 2002.

[2] Y. Uny Cao, A. S. Fukunaga, and A. B. Kahng. Cooperative mobile robotics: Antecedents and directions. *Autonomous Robots*, 4(1):7–23, March 1997.

[3] M. Cieliebak, P. Flocchini, G. Prencipe, and N. Santoro. Solving the robots gathering problem. In *30th Int. Colloq. on Automata, Languages and Programming*, pages 1181–1196, 2003.

[4] M. Cieliebak and G. Prencipe. Gathering autonomous mobile robots. In *Proc. 9th Int. Colloq. on Structural Information and Communication Complexity*, pages 57–72, June 2002.

[5] M. Erdmann and T. Lozano-Perez. On multiple moving objects. In *Proc. IEEE Conf. on Robotics and Automation*, pages 1419–1424, April 1986.

[6] P. Flocchini, G. Prencipe, N. Santoro, and P. Widmayer. Distributed coordination of a set of autonomous mobile robots. In *IEEE Intelligent Vehicles Symposium (IV 2000)*, pages 480–485, October 2000.

[7] P. Flocchini, G. Prencipe, N. Santoro, and P. Widmayer. Gathering of autonomous mobile robots with limited visibility. In *Proc. 18th Symp. on Theoretical Aspects of Computer Science*, pages 247–258, February 2001.

[8] Y. Kuniyoshi, S. Rougeaux, M. Ishii, N. Kita, S. Sakane, and M. Kakikura. Cooperation by observation - the framework and basic task patterns. In *Proc. Int. Conf. on Robotics and Automation*, pages 767–774, 1994.

[9] L. Parker, C. Touzet, and F. Fernandez. Techniques for learning in multi-robot teams. In T. Balch and L. Parker, editors, *Robot Teams: From Diversity to Polymorphism.* A. K. Peters, 2001.

[10] G. Prencipe. CORDA: Distributed coordination of a set of atonomous mobile robots. In *Proc. 4th European Research Seminar on Advances in Distributed Systems*, pages 185–190, May 2001.

[11] G. Prencipe. Instantaneous actions vs. full asynchronicity: Controlling and coordinating a set of autonomous mobile robots. In *Proc. 7th Italian Conf. on Theoretical Computer Science*, pages 185–190, October 2001.

[12] G. Prencipe. *Distributed Coordination of a Set of Atonomous Mobile Robots.* PhD thesis, Universita Degli Studi Di Pisa, 2002.

[13] K. Sugihara and I. Suzuki. Distributed algorithms for formation of geometric patterns with many mobile robots. *Journal of Robotic Systems*, 13(3):127–139, 1996.

[14] I. Suzuki and M. Yamashita. Agreement on a common x-y coordinate system by a group of mobile robots. In *Proc. Dagstuhl Seminar on Modeling and Planning for Sensor-Based Intelligent Robots*, September 1996.

[15] I. Suzuki and M. Yamashita. Distributed anonymous mobile robots - formation and agreement problems. In *Proc. 3rd Colloq. on Structural Information and Communication Complexity*, pages 313–330, 1996.

[16] I. Suzuki and M. Yamashita. Distributed anonymous mobile robots: Formation of geometric patterns. *SIAM J. on Computing*, 28:1347–1363, 1999.

[17] R. Wenger. Helly-type theorems and geometric transversals. In Jacob E. Goodman and Joseph O'Rourke, editors, *Handbook of Discrete and Computational Geometry*, chapter 4, pages 63–82. CRC Press LLC, 1997.

[18] D. Yoshida, T. Masuzawa, and H. Fujiwara. Fault-tolerant distributed algorithms for autonomous mobile robots with crash faults. *Systems and Computers in Japan*, 28:33–43, 1997.