A Distributed Algorithm for Gathering Many Fat Mobile Robots in the Plane

Chrysovalandis Agathangelou  Chryssis Georgiou  Marios Mavronicolas
Department of Computer Science
University of Cyprus
CY-1678 Nicosia, Cyprus

Abstract

In this work we consider the problem of gathering autonomous robots in the plane. In particular, we consider non-transparent unit-disc robots (i.e., fat) in an asynchronous setting. Vision is the only mean of coordination. Using a state-machine representation we formulate the gathering problem and develop a distributed algorithm that solves the problem for any number of robots.

The main idea behind our algorithm is for the robots to reach a configuration in which all the following hold: (a) The robots’ centers form a convex hull in which all robots are on the convex, (b) Each robot can see all other robots, and (c) The configuration is connected, that is, every robot touches another robot and all robots together form a connected formation. We show that starting from any initial configuration, the robots, making only local decisions and coordinate by vision, eventually reach such a configuration and terminate, yielding a solution to the gathering problem.

1 Introduction

Motivation and Prior Work: There is an increasing number of applications that could benefit from having a team of autonomous robots cooperate and complete various tasks in a self-organizing manner. These tasks could, for example, require the robots to work in dangerous and harsh environments (e.g., for space, underwater or military purposes) or require high accuracy or speed (e.g., in nanotechnology, scientific computing) or be of research interest (e.g., artificial intelligence). It is usually desirable for the robots to be as simple and cheap as possible and have limited computing power, in order to be able to produce them fast in large numbers.

A fundamental problem that has drawn much attention in the recent years is Gathering [2, 1, 11, 4, 13], where a team of autonomous mobile robots must gather to a certain point or region or form a certain formation (e.g., geometric shapes) in the plane. The problem has been studied under various modeling assumptions. For example, asynchronous, semi-synchronous and synchronous settings have been considered. Robots may have a common coordination system or have common sense of direction and use compasses to navigate in the plane, may have stable memory or be history oblivious. A modeling feature that is shared by all prior works considering the asynchronous setting is that robots are equipped with a vision device (e.g., a camera) and operate under the so called Look-Compute-Move cycle. Within a cycle, a robot takes a snapshot of the plane (Look), based on the snapshot it might perform some local computations (Compute), and it might decide to move to some point in the plane (Move). The range of visibility of robots may be limited or unlimited. We refer the reader to Surveys [4, 13] and the recent monograph [10] for a more comprehensive exposition of works on the gathering problem.

Up until the work of Czyzowicz et al. [8], the gathering problem was considered only under the assumption that robots are a point on the plane and are transparent, that is, a robot can see through another robot.
These assumptions do not reflect reality, as real robots are not points, but instead they have a physical extent. Furthermore, robots are not transparent, that is, robots may block the view of other robots or robots may collide. Having this in mind, Czyzowicz et al. [8] initiated the study of the gathering problem with fat robots, that is, non-transparent unit-disks (disks of radius of 1 unit). As fat robots cannot occupy the same space on the plane, the gathering problem can no longer require robots to gather at the same point. Instead, per [8], gathering fat robots means forming a configuration for which the union of all discs representing them is connected. The model considered in [8] is the following: Robots operate in Look-Compute-Move cycles, they are identical, anonymous, history oblivious, non-transparent, and fat. They do not share a common coordination system and the only means of coordination is by vision; robots have unlimited visibility, unless their view is obstructed by another robot. An asynchronous setting is considered, modeled by an adaptive adversary that can stop any robot for finite time, control the “speed” of any robot or cause robots moving into intersecting trajectories to collide. Under this model, the authors present solutions for the gathering problem for three and four robots. The proposed solutions consider exhaustively all possible classes of configurations in which robots may be found; a different gathering strategy corresponds to each possible case. As this approach cannot be generalized for larger number of robots (the cases grow exponentially as the number of robots increases), the authors left open the question of whether it is possible to solve gathering for any collection of \( n \geq 5 \) fat robots.

Our Contribution: In this work we provide a positive answer to the above question. In particular, we consider the model of [8] with the additional assumption of chirality [10] (i.e., robots agree on the orientation of the axes of their local coordinator system) and present a distributed algorithm for the gathering problem for any number \( n \) of fat robots.

The key feature of our solution is to bring the robots in a configuration of full visibility where all robots can see all other robots. However, the power of the adversary and the fact that robots are non-transparent makes this task challenging. We have overcome this challenge by requiring robots to aim in forming a convex hull in which all robots will be on the convex. During the computation, robots that are on the convex do not move and robots that are inside the convex hull try to move on the convex hull. However, if robots that are on the convex hull realize that they obstruct other robots that are also on the convex hull from seeing each other, then they move with direction outside of the convex hull in such a way, that they no longer cause any obstruction of the view of the other robots. Furthermore, if robots on the convex hull realize that there is no “enough space” for robots that are inside the convex to be placed on the convex, they move to a direction outside of the convex to make space. All these are further complicated due to asynchrony, as robots may have very different local views of the system. We show that eventually the convex hull will “expand” in such a way that all robots will be on the convex hull and no three robots will be on the same line. This leads to a configuration that all robots have full visibility. This is the first conceptual phase of the algorithm.

In the second conceptual phase of the algorithm, once all robots have full visibility and they are aware of this, robots start to converge in such a manner that full visibility is not lost (again, asynchrony complicates issues). To do so, robots exploit their knowledge of \( n \) and of the common unit of distance (since all robots are unit-disks, this gives them “for free” a common measure of distance [8]). We show that eventually all robots form a connected configuration and terminate, yielding a solution to the gathering problem.

The key in successfully proving the correctness of the algorithm is the formulation of the model and the problem using a state-machine representation. This enabled us to employ typical techniques for proving safety and liveness properties and argue on the state transitions of the robots, which against asynchrony it can be a very challenging task.

Other Works Considering Fat Robots. After the work in [8] some attempts were made in solving the gathering problem with \( n \geq 5 \) fat robots. However these works consider different models than the one
considered in [8]. In [6] it is assumed that the fat robots are transparent. This assumption makes the problem significantly easier, as robots have full visibility at all times. As discussed above, having the robots reach a configuration with full visibility was the main challenge in our work. In [7] fat robots are non-transparent and have limited visibility, but a synchronous setting is considered. Furthermore, the gathering point is predefined (given as an input to the robots) and the goal is for the robots to gather in an area as close as possible to this point. Two versions of the problem are studied: in continuous space and time, and in discrete space (essentially $\mathbb{Z}^2$) and time. In the continuous case a randomized solution is proposed. In the discrete case the proposed solutions require additional modeling assumptions such as unique robot ids, or direct communication between robots, or randomization. The work in [9] also considers fat robots with limited visibility, but in an asynchronous setting. In contrast with the model we consider, robots have a common coordination system, that is they agree both on a common origin and axes (called Consistent Compass in [10]). The objective of the robots is to gather to a circle with center C, which is given as an input along with the radius of the circle. The common coordination system and the predefined knowledge of the circle to be formed enables the use of geometric techniques that cannot be used in our model. In [5] they consider fat robots with limited visibility and without a common coordination system, but in a synchronous setting. Furthermore the correctness of the proposed algorithm is not proven analytically, but rather demonstrated via simulations.

## 2 Model and Definitions

Our model of computation is a formalization of the one presented in [8] (with the additional assumption of chilarity); our formalism follows the one from [3].

**Robots:** We assume $n$ asynchronous fault-free robots that can move in straight lines on the (infinite) plane. The robots are fat [8]: they are closed unit discs. They are identical, anonymous and indistinguishable. They do not have access to any global coordination system, but we assume that the robots agree on the orientation of the axes (i.e., per [10] they have chilarity\(^1\)). Robots are equipped with a 360-degree-angle vision devise (e.g., camera) that enables the robots to take snapshots of the plane. The vision devise can capture any point of the plane (has unlimited range) provided there is no obstacle (e.g., another robot). We assume that robots know $n$.

**Geometric configuration:** A geometric configuration is a vector $\mathbf{G} = (c_1, c_2, \ldots, c_n)$ where each $c_i$ represents the center of the position of robot $r_i$ on the plane. Informally speaking, a configuration can be viewed as a snapshot of the robots on the plane. Note that the fact that robots are fat prohibits the formation of a configuration in which any two robots share more than a point (on the perimeter of their unit discs) in the plane.

We say that a geometric configuration $\mathbf{G}$ is **connected**, if between any two points of any two robots there exists a polygonal line each of whose points belongs to some robot. Informally, a configuration is connected if every robot touches another robot (i.e., their circles representing the robots are tangent) and all robots together form a connected formation.

**Visibility and fully visible configuration:** We say that point $p$ in the plain is **visible** by a robot $r_i$ (or equivalently, $r_i$ can see $p$) if there exists a point $p_i$ in the circle bounding robot $r_i$ such that the straight segment $(p_i, p)$ does not contain any point of any other robot. From this it follows that a robot $r_i$ can see another robot $r_j$ if there exists at least one point on the bounding circle of $r_j$ that is visible by $r_i$.

\(^1\)In other words, we assume that they have a common understanding of what is left or right. Note this is a weaker assumption than having a common coordinate system or having a compass [10].
Given a geometric configuration $G$, if a robot $r_i$ can see all other robots, then we say that robot $r_i$ has full visibility in $G$. If all robots have full visibility in $G$, then we say that configuration $G$ is fully visible.

**Robots’ states:** Formally, each robot $r_i$ is modeled as a (possibly infinite) state machine with state set $S_i$; $i$ is the index of robot $r_i$ (used only for reference purposes). Each set $S_i$ contains five states: **Wait**, **Look**, **Compute**, **Move**, and **Terminate**. Initially each robot is in state **Wait**. State **Terminate** is a terminating state: once a robot reaches this state it does not take any further steps. We now describe each state:

- In state **Wait**, robot $r_i$ is idling. In addition, the robot has no memory of the steps occurred prior entering this state (that is, every time a robot gets into state **Wait**, it looses any recollection of past steps – robots are history oblivious).

- In state **Look**, robot $r_i$ takes a snapshot of the plane and identifies the robots that are visible to it. We denote by $V_i$ the set of the centers of the robots that are visible to robot $r_i$ when it takes a snapshot in configuration $G$. That is, $V_i \subseteq G$ is the local view of robot $r_i$ in configuration $G$. Note that this view does not change in subsequent configurations unless the robot takes a new snapshot. In a nutshell, in this state, the robot takes as an input a configuration $G$ and outputs the local view $V_i \subseteq G$.

- In state **Compute**, robot $r_i$ runs a local algorithm, call it $A_i$, that takes as an input the local view $V_i$ (that is, the output of the previous state **Look**) and outputs a point $p$ in the plane. This point is specified from $V_i$, hence we will write $p = A_i(V_i)$. If $A_i$ outputs the special point $\bot$, then the robot’s state changes into state **Terminate**. Otherwise it changes into state **Move** (intuitively, in this case $p$ is the point that the center of the robot will move to). Note that it is possible for $p = c_i$, that is, the robot might decide not to move.

- In state **Move**, robot $r_i$ starting from its current position, called start point, moves on a straight line towards point $A_i(V_i)$ (as calculated in state **Compute**). We call $A_i(V_i)$ the target point of $r_i$. If during the motion the robot touches some other robot (i.e., the circles representing these robots become tangent) it stops and the robot’s state changes into state **Wait**. As we discuss next, the adversary may also stop a robot at any point before reaching its target point. Again, in this case, the robot’s state changes into state **Wait**. If the robot finds no obstacles or it is not stopped by the adversary then it eventually reaches its target point (its center is placed on $A_i(V_i)$) and its state changes into state **Wait**.

**State configuration:** A state configuration is a vector $S = (s_1, s_2, \ldots, s_n)$ where each $s_i$ represents the state of robot $r_i$. An initial state configuration is a configuration $S$ in which each $s_i$ is an initial state of robot $r_i$ (that is, $\forall i \in [1, n], s_i = \text{Wait}$). Similarly, a terminal state configuration is a configuration $S$ in which each $s_i$ is a terminating state of robot $r_i$ (that is, $\forall i \in [1, n], s_i = \text{Terminate}$).

**Robot configuration:** A robot configuration is a vector $R = ((s_1, c_1), \ldots, (s_n, c_n))$ where each pair $(s_i, c_i)$ represents the state of robot $r_i$ and the position of its center on the plane. (Informally, a robot configuration is the combination of a geometric configuration with the corresponding state configuration.)

**Adversary and events:** We model asynchrony as events caused by an online and omniscient adversary. The adversary can control the speeds of the robots, it can stop moving robots, and it may cause moving robots to collide, provided that their trajectories have an intersection point. Specifically, we consider the following events (state transitions):

- **Look($r_i$):** This event causes robot $r_i$ that is in state **Wait** to get into state **Look**.
Compute($r_i$): This event causes robot $r_i$ that is in state Look to get into state Compute.

Done($r_i$): This event causes robot $r_i$ that is in state Compute and its local algorithm $A_i$ has returned the special point $\bot$, to get into the terminating state Terminate.

Move($r_i$): This event causes robot $r_i$ that is in state Compute and its local algorithm $A_i$ has returned a point other than $\bot$, to get into state Move.

Stop($r_i$): This event causes robot $r_i$ that is in state Move to get into state Wait. Robot $r_i$ is stopped at some point in the straight segment between its start point and its target point $A_i(V_i)$ (under a constraint discussed next).

Collide($R$): This event causes a subset of the robots $R$ that are in state Move and their trajectories have an intersecting point to collide (i.e., their circles representing the robots become tangent). Note that $2 \leq |R| \leq n$ (two or more robots could collide between them but only one collision occurs per a Collide event). Also, other robots that are in state Move could be stopped (without colluding with other robots). All affected robots are now in state Wait.

Arrive($r_i$): This event causes robot $r_i$ that is in state Move to arrive at its target point and change its state into Wait.

Note that events Look($r_i$), Move($r_i$), Stop($r_i$) and Arrive($r_i$) may also cause robots (other than $r_i$) that are in state Move to remain in that state, but on a different position on the plane (along their trajectories, and closer to their destination).

Figure 1 depicts a cycle of the state transitions of a robot $r_i$; it is understood that for event collide($R$), $r_i \in R$.

Execution: An execution fragment is an alternating sequence of robot configurations and events. Formally, an execution fragment $\alpha$ is a (finite or infinite) sequence of $R_0, e_1, R_1, e_2, \ldots$, where each $R_k$ is a robot configuration and each $e_k$ is an event. If $\alpha$ is finite, then it ends in a configuration. An execution is an execution fragment where $R_0$ is an initial configuration.

Liveness conditions: We impose the following liveness conditions (they are basically restrictions on the adversary):

1. In an infinite execution, each robot may take infinitely many steps.
2. During a Move event, each robot traverses at least a distance $\delta > 0$ unless its target point is closer than $\delta$. Formally, each robot $r_i$ traverses at least a distance $\min\{\text{dist}_i(\text{start}, \text{target}), \delta\}$, where $\text{dist}_i(\text{start}, \text{target})$ denotes the distance between the start and target points of robot $r_i$. Parameter $\delta$ is not known to the robots (or to their local algorithms).

Gathering: We now state the problem we consider in this work:

Definition 1 (Gathering problem) In any execution, there is a connected, fully visible, terminal robot configuration.

3 Geometric Functions

In this section we present a collection of functions that perform geometric calculations. These functions are used by the robots’ local algorithm as shown in the next section. In this section we present these functions in a general manner, with reference to the centers of unit discs on the plane (that is, not necessarily for robots). After the presentation of each function, we give some insight on how this function is used by the robots’ local algorithm.

3.1 Function On-Convex-Hull

We denote by $CH(c_1, c_2, \ldots, c_m)$ the convex hull formed by points $c_1, c_2, \ldots, c_m$, and by $onCH(c_1, c_2, \ldots, c_m) \subseteq \{c_1, c_2, \ldots, c_m\}$ the set of points that are on the convex hull. Then, function On-Convex-Hull solves the following algorithmic problem:

**Input:** A set of $m$ points $c_1, c_2, \ldots, c_m$ and an additional point $c$.

**Output:** YES if $c \in onCH(c_1, c_2, \ldots, c_m)$ otherwise NO.

Function On-Convex-Hull involves the computation of the convex hull formed by points $c_1, c_2, \ldots, c_m$ and a check whether $c$ is one of the points on the convex hull. The function returns, besides YES or NO, also set $onCH(c_1, c_2, \ldots, c_m)$. This function can easily be implemented using, for example, Graham’s Convex Hull Algorithm [12].

Insight: This function is called by a robot $r$ with center $c$. The $m \leq n$ points are the centers of the robots that robot $r$ can see (its local view) in the current configuration. The robot wishes to check whether its center belongs on the convex hull formed by the centers of the robots in its local view. In the case of full visibility ($m = n$) robot $r$ can check whether itself as well as all other robots are on the convex hull. If this is the case, then by the definition of full visibility and of the convex hull, all robots can potentially see all other robots.

3.2 Function Move-to-Point

Function Move-to-Point solves the following algorithmic problem:

**Input:** Two points $c_1$ and $c_2$ and a positive integer $m$.

**Output:** Point $\mu$ defined as follows: Consider the straight segment $\overline{c_1c_2}$ and let $\overline{pc_2}$ be the straight segment which is vertical to $\overline{c_1c_2}$ and $p$ is on the perimeter of the unit disc with center $c_2$ and with direction inside of
the convex hull. Next consider the point $c$ on segment $pc_2$ which has distance $\frac{1}{2m} - \epsilon$ from $c_2$. Then point $\mu$ is the intersection of the straight segment $c_1\mu$ and the perimeter of the unit disc with center $c_2$. See Figure 2 for an example.

It is not difficult to see that Function Move-to-Point involves simple geometric calculations.

**Insight:** This function is called by robot $r$ with center $c_1$. Point $c_2$ corresponds to the center of a robot that $r$ wants to touch (i.e., the unit discs representing the robots become tangent). For this purpose robot $r$ must move towards the other robot in such a way that their circles become tangent at point $\mu$. As we will see later, this function is called with $m = n$, $n$ being the number of robots in the system. Intuitively, distance $\frac{1}{2m} - \epsilon$ is used to aid robot $r$ to remain visible by other robots (that is, it will not be hidden by the robot with center $c_2$.)

### 3.3 Function Find-Points

Function Find-Points solves the following algorithmic problem:

**FIND POINTS**

**Input:** Given a Convex Hull, let points $c_1, c_2, \ldots, c_m$ be the points that are on the convex hull out of the total $n$ points.

**Output:** A set of $k < m$ points $p_1, \ldots, p_k$ that a unit disc with center $p_i$, $1 \leq i \leq k$, can be placed on the convex hull without causing the convex hull to change. (It is possible that $k = 0$.)

We now give the details of function Find-Points.

**Function: Find-Points**

**Set** $Points = \emptyset$;

**Consider** the points on the convex hull with a clockwise ordering;

**For** each pair $c_l, c_r$ of neighboring points on the convex hull **do**:

**If** the length of line $c_lc_r$ is greater than or equal to 2, **then**

Let $\mu$ be the center of $c_lc_r$;

Draw a vertical line on $c_lc_r$, that starts from point $\mu$ and moves outside the convex, until distance $\frac{1}{n}$ from $\mu$. Let $p$ be the ending point;

Let $c_{l-1}$ be the left neighbor of $c_l$ and $c_{r+1}$ be the right neighbor of $c_r$ (mods are omitted for simplicity of notation);

Consider the straight segment that starts from $c_{l-1}$ and goes through $c_l$ and the straight segment that starts from $c_{r+1}$ and goes through $c_r$. Let $t$ be the point where the two segments intersect;

Consider the unit disc formed with center $p$;

**If** no point of this unit disc is above or $p$ has distance $\frac{1}{n}$ or more from the line segments $c_{l-1}t$ and $tc_{r+1}$, **then** $Points = Points \cup \{p\}$; (See Figure 3 for an example.)

**Return** $Points$;
**Insight:** This function is called by a robot that is not on the convex hull and wishes to see whether there is at least one point that it could move and get on the convex hull without causing the convex hull to change. The number of the input points, $m$ is smaller than $n$ i.e. $m < n$, because if this function is called, it means that at least one robot is not on the convex hull. Given a line segment $\overline{\mu c}$ of length at least 2, a simple solution would be for the robot to move in the middle of this line; however, that would cause robots $r_1$ and $r_c$ not to be visible to each other, which is another property we wish to have (all robots on the convex hull must be able to see each other). Therefore, we check whether the robot could be placed at some vertical distance away from the middle, so that $r_1$ and $c_r$ can still see other, but at the same time the tangents on the convex hull are not affected (in which case it would cause the convex hull to change).

**Lemma 1** If a unit disk is placed on $onCH(V_i)$ with center a point that was returned by Function $FindPoints$, it will not cause $onCH(V_i)$ to change.

**Proof:** The correctness follows by close investigation of the code of Function $FindPoints$. If a unit disk moves to one of the points returned, it will not cause $onCH(V_i)$ to change provided that other unit disks do not move.

**Lemma 2** Given a convex hull, for any two adjacent unit disks with centers $c_l$ and $c_r$ on the convex hull there exists a minimum distance between $c_l$, $c_r$ for which Function $FindPoints$ would return a point between $c_l$, $c_r$. We refer to this distance as the safe distance.

**Proof:** Consider that given a number of points, a convex hull always exists. Consider four neighbor points on a convex hull, as shown on Figure 3, without loss of generality. In order for a unit disk with center $p$ to be on the convex hull and not cause the current convex hull to change, the distance between $\mu$, the middle point of $\overline{c_l c_r}$ and $p$ must be at least $\frac{1}{n}$. Note that $p$ is outside of the current convex hull. Additionally, consider $q$ the point on the line segment $\overline{pc_{r+1}}$, where a vertical line to $c_r$ starts from line segment $\overline{pc_{r+1}}$ with direction to the inside of the convex hull. Then $d(q, c_r)$ must be equal with at least $\frac{1}{n}$, where $r$ is the point that $\overline{pc_{r+1}}$ is tangent with $\overline{pc_r}$. Angle $\overline{pr\mu}$ is equal with angle $\overline{c_r rq}$

We need to calculate the distance between $c_l$ and $c_r$ which will give as the safe distance. The distance between $c_l$ to $\mu$ must be equal with the distance between $c_r$ to $\mu$. We need to calculate both $d(\mu, c_r)$ and $d(\mu, c_l)$, find the biggest and double it, in order to find the safe distance. First we must calculate the necessary distance between $\mu$ and $c_r$.

Observe that $d(\mu, c_r) = d(\mu, r) + d(r, c_r)$. Firstly we calculate $d(\mu, r)$ We have that $\tan(\overline{pr\mu}) = \frac{1}{d(\mu, r)}$, hence $d(\mu, r) = \frac{1}{n \cdot \tan(\overline{pr\mu})}$ Secondly we calculate $d(r, c_r)$, we have that $\sin(\overline{c_r rq}) = \frac{1}{d(r, c_r)}$, hence $d(r, c_r) = \frac{1}{n \cdot \sin(\overline{c_r rq})} = \frac{1}{n \cdot \sin(\overline{pr\mu})}$

Now we are ready to calculate $d(\mu, c_r)$, it follows that $d(\mu, c_r) = \frac{1}{n \cdot \tan(\overline{pr\mu}) + \frac{1}{n \cdot \sin(\overline{pr\mu})}}$

This is the minimum distance that $\overline{pc_r}$ must be. We do the same as above with $\overline{pc_l}$ and choose the biggest distance between the two, double it and set it as safe distance.

**3.4 Function Connected-Components**

Consider a set of $m$ unit discs on the plane. A connected component of this set consists all unit discs that are connected (between any two points of any two unit discs there exists a polygonal line each of whose points belong to some unit disc). In a connected component there can be up to two empty spaces of distance less or equal to $1/2m$. If there are more than two such spaces, then this component is considered as another connected component. Note that a given set of unit discs may contain many connected components and only one in the case that all unit discs are connected.
**High level idea:** A component part consists of unit disks that are tangent and there exists a polygonal line each of whose points belong to a robot. A component can have one, two or three component parts with maximum space between adjacent parts $\frac{1}{2m}$. This is a simple problem but with many possible cases that can be seen on the code of this function. We now proceed with the full code of the function.

Function `Connected-Components` solves the following algorithmic problem:

**CONNECTED COMPONENTS**

**Input:** A set of $m$ points $c_1, c_2, \ldots, c_m$ and an additional point $c$.

**Output:** A set of pairs of the form $\langle (c_l, c_r), k \rangle$. Each pair $(c_l, c_r)$ represents a connected component of unit discs, where $c_l$ is the center of the leftmost unit disc and $c_r$ the center of the rightmost unit disc in the component. $k$ is the number of unit discs contained in this component (including $c_l$ and $c_r$).

We now give the details of function `Connected-Components`. The correctness of the function (that is, the proof that it correctly solves the above problem) follows by close investigation of the code of the function. We use the notation $\langle (c_{rx}, c_{ly}), k_{rx,ly} \rangle$, to denote the component where $c_{rx}$ is the center of the first unit disc on the right of $space_{x}$, $c_{ly}$ is the center of the last unit disc on the left of $space_{y}$, and $k_{rx,ly}$ is the number of unit discs between (and including) $c_{rx}$ and $c_{ly}$; $x, y, k_{rx,ly}$ are positive integers.

**Function:** `Connected-Components`

**Set** initial = $c$ and **Components** = $\emptyset$;

Starting from initial, move to the right along connected unit discs until a space is reached. Call this space, $space_{0}$;

1. **If** the length of $space_{0}$ is less than or equal to $\frac{1}{2m}$ **then** continue moving to the right along connected unit discs until another space is reached. Call this space, $space_{1}$;

(a) **If** the length of $space_{1}$ is less than or equal to $\frac{1}{2m}$ **then** continue moving to the right along connected unit discs until another space is reached. Call this space, $space_{2}$;

i. **If** the length of $space_{2}$ is less than or equal to $\frac{1}{2m}$ **then** continue moving to the right along connected unit discs until another space is reached. Call this space, $space_{3}$;

**Set** **Components** = **Components** $\cup \langle (c_{r2}, c_{l3}), k_{r2,l3} \rangle$ $\cup \langle (c_{r1}, c_{l2}), k_{r1,l2} \rangle$;

From initial, move to the left along connected unit discs until a space is reached. Call this space, $space_{4}$;

**Set** **Components** = **Components** $\cup \langle (c_{r4}, c_{l0}), k_{r4,l0} \rangle$ $\cup \langle (c_{r0}, c_{l1}), k_{r0,l1} \rangle$;

**Set** initial = $c_{r3}$ (center of first unit disc on the right of $space_{3}$).

**If** the unit disc with center $c$ (the input’s additional point) is included in one of the newly included components **and** this is not the first iteration of the procedure, **then remove** mul-

---

*Figure 3: An example where point $p$ is not valid and hence it will not be included in set Points.*
tiplcities and terminate;
Else repeat procedure with (new) initial.

ii. If the length of space2 is greater than \( \frac{1}{2m} \) then from initial, move to the left along connected unit discs until a space is reached. Call this space, space3;

A. If the length of space3 is greater than \( \frac{1}{2m} \) then Set Components = Components ∪ \( \langle (c_{r3}, c_{l2}), k_{r3,l2} \rangle \);
Set initial = \( r_{l3} \).
If the unit disc with center \( c \) (the input’s additional point) is included in the newly included component and this is not the first iteration of the procedure, then remove multiplicities and terminate;
Else repeat procedure with (new) initial.

B. If the length of space3 is less than or equal to \( \frac{1}{2m} \) then Set Components = Components ∪ \( \langle (c_{r0}, c_{l1}), k_{r0,l1} \rangle \) ∪ \( \langle (c_{r1}, c_{l2}), k_{r1,l2} \rangle \);
continue moving to the left along connected unit disks until another space is reached.
Call this space, space4; Set Components = Components ∪ \( \langle (c_{r4}, c_{l3}), k_{r4,l3} \rangle \) ∪ \( \langle (c_{r3}, c_{l0}), k_{r3,l0} \rangle \);
Set initial = \( r_{l3} \).
If the unit disc with center \( c \) (the input’s additional point) is included in the newly included component and this is not the first iteration of the procedure, then remove multiplicities and terminate;
Else repeat procedure with (new) initial.

(b) If the length of space1 is greater than \( \frac{1}{2m} \) then from initial, move to the left along connected unit discs until a space is reached. Call this space, space2;

i. If the length of space2 is greater than \( \frac{1}{2m} \) then Set Components = Components ∪ \( \langle (c_{r2}, c_{l1}), k_{r2,l1} \rangle \);
Set initial = \( r_{l1} \).
If the unit disc with center \( c \) (the input’s additional point) is included in the newly included component and this is not the first iteration of the procedure, then remove multiplicities and terminate;
Else repeat procedure with (new) initial.

ii. If the length of space2 is less than or equal to \( \frac{1}{2m} \) then continue moving to the left along connected unit discs until another space is reached. Call this space, space3;

A. If the length of space3 is greater than \( \frac{1}{2m} \) then Set Components = Components ∪ \( \langle (c_{r3}, c_{l1}), k_{r3,l1} \rangle \);
Set initial = \( r_{l1} \).
If the unit disc with center \( c \) (the input’s additional point) is included in the newly included component and this is not the first iteration of the procedure, then remove multiplicities and terminate;
Else repeat procedure with (new) initial.

B. If the length of space3 is less than or equal to \( \frac{1}{2m} \) then continue moving to the left along connected unit discs until another space is reached. Call this space, space4;
Set Components = Components ∪ \( \langle (c_{r0}, c_{l1}), k_{r0,l1} \rangle \) ∪ \( \langle (c_{r2}, c_{l0}), k_{r2,l0} \rangle \) ∪ \( \langle (c_{r3}, c_{l2}), k_{r3,l2} \rangle \) ∪ \( \langle (c_{r4}, c_{l3}), k_{r4,l3} \rangle \);
Set initial = \( c_{r1} \).
If the unit disc with center \( c \) (the input’s additional point) is included in the newly included component and this is not the first iteration of the procedure, then remove
multiplicities and terminate;
Else repeat procedure with (new) initial.

2. If the length of $space 0$ is greater than $\frac{1}{2m}$ then from initial, move to the left along connected unit discs until a space is reached. Call this space, $space 1$;

(a) If the length of $space 1$ is greater than $\frac{1}{2m}$ then Set Components = Components $\cup$ $\langle (c_1, c_0), k_{r1,0} \rangle$;
Set initial = $c_0$.
If the unit disc with center $c$ (the input’s additional point) is included in the newly included component and this is not the first iteration of the procedure, then remove multiplicities and terminate;
Else repeat procedure with (new) initial.

(b) If the length of $space 1$ is less than or equal to $\frac{1}{2m}$ then continue moving to the left along connected unit discs until another space is reached. Call this space, $space 2$;

i. If the length of $space 2$ is greater than $\frac{1}{2m}$ then Set Components = Components $\cup$ $\langle (c_2, c_0), k_{r2,0} \rangle$;
Set initial = $c_0$.
If the unit disc with center $c$ (the input’s additional point) is included in the newly included component and this is not the first iteration of the procedure, then remove multiplicities and terminate;
Else repeat procedure with (new) initial.

ii. If the length of $space 2$ is less than or equal to $\frac{1}{2m}$ then continue moving to the left along connected unit discs until another space is reached. Call this space, $space 3$;

A. If the length of $space 3$ is greater than $\frac{1}{2m}$ then Set Components = Components $\cup$ $\langle (c_3, c_0), k_{r3,0} \rangle$;
Set initial = $c_0$.
If the unit disc with center $c$ (the input’s additional point) is included in the newly included component and this is not the first iteration of the procedure, then remove multiplicities and terminate;
Else repeat procedure with (new) initial.

B. If the length of $space 3$ is less than or equal to $\frac{1}{2m}$ then continue moving to the left along connected unit discs until another space is reached. Call this space, $space 4$;
Set Components = Components $\cup$ $\langle (c_{r1}, c_0), k_{r1,0} \rangle \cup \langle (c_{r2}, c_1), k_{r2,1} \rangle \cup \langle (c_{r3}, c_2), k_{r3,2} \rangle \cup \langle (c_{r4}, c_3), k_{r4,3} \rangle$;
Set initial = $c_r$.
If the unit disc with center $c$ (the input’s additional point) is included in the newly included component and this is not the first iteration of the procedure, then remove multiplicities and terminate;
Else repeat procedure with (new) initial.

---

**Insight:** This function is called by a robot $r$ with center $c$. The $m$ points are the centers of the robots that robot $r$ can see (its local view) in the current configuration. As we will see later, this function is called when the robot can see all other robots, i.e., $m = n$. The robot wishes to find the connected components formed in the current configuration. Intuitively, we can include two spaces of length $1/2n$ in a configuration, since if all the robots can see each other, then the robots can move taking steps of length $1/2n$ until they meet.
3.5 Function How-Much-Distance

Function How-Much-Distance solves the following algorithmic problem:

| HOW MUCH DISTANCE |
|-------------------|
| **Input:** A set of $m$ points $c_1, c_2, \ldots, c_m$ and an additional point $c$. |
| **Output:** One of the numbers 1, 2 or 3. Consider the connected components formed by the unit discs with centers $c_1, c_2, \ldots, c_m$. If the unit disc with center $c$ is the rightmost (the straight direction is considered to be the inside of the convex hull) element of the component that has the smallest (space-wise) distance between the components, then the answer is 1. If all components have the same distance, then the answer is 2. Otherwise the answer is 3. |

Function How-Much-Distance calls function Connected-Components (Section 3.4) to get the connected components formed by the unit discs with centers $c_1, c_2, \ldots, c_m$. Then it checks the distances between the components and returns 1, 2 or 3 accordingly.

**Insight:** This function is called by robot $r$ with center $c$. The robot wants to check whether it is the rightmost robot in the connected component with the smallest distance among the components formed by the robots in its local view.

3.6 Function In-Largest-Component

Function In-Largest-Component solves the following algorithmic problem:

| IN LARGEST COMPONENT |
|-----------------------|
| **Input:** A set of $m$ points $c_1, c_2, \ldots, c_m$ and an additional point $c$. |
| **Output:** One of the numbers 1, 2 or 3. Consider the connected components formed by the unit discs with centers $c_1, c_2, \ldots, c_m$. If the unit disc with center $c$ belongs in the largest component (wrt the number of discs), then the answer is 1; if all the components are larger than the one it belongs, then the answer is 2. Otherwise the answer is 3. |

Function In-Largest-Component calls function Connected-Components (Section 3.4) to get the connected components formed by the unit discs with centers $c_1, c_2, \ldots, c_m$. Then it checks the sizes of the components and where the unit disc with center $c$ belongs to, and returns 1, 2 or 3 accordingly.

**Insight:** This function is called by robot $r$ with center $c$. The robot wants to check whether it belongs in the largest component among the components formed by the robots in its local view.

3.7 Function In-Smallest-Component

Function In-Smallest-Component solves the following algorithmic problem:

| IN SMALLEST COMPONENT |
|-----------------------|
| **Input:** A set of $m$ points $c_1, c_2, \ldots, c_m$ and an additional point $c$. |
| **Output:** One of the numbers 1, 2 or 3. Consider the connected components formed by the unit discs with centers $c_1, c_2, \ldots, c_m$. If the unit disc with center $c$ belongs in the smallest component (wrt the number of discs), then the answer is 1; if all the components are smaller than the one it belongs, then the answer is 2. Otherwise the answer is 3. |
Function \textit{In-Smallest-Component} calls function \textit{Connected-Components} (Section 3.4) to get the connected components formed by the unit discs with centers $c_1, c_2, \ldots, c_m$. Then it checks the sizes of the components and where the unit disc with center $c$ belongs to, and returns 1, 2 or 3 accordingly.

\textbf{Insight:} This function is called by robot $r$ with center $c$. The robot wants to check whether it belongs in the smallest component among the components formed by the robots in its local view.

\subsection*{3.8 Function \textit{In-Straight-Line-2}}

Function \textit{In-Straight-Line-2} solves the following decision problem:

\begin{itemize}
\item \textbf{Input:} A set of 3 points $c_l, c_m$ and $c_r$.
\item \textbf{Output:} \textit{YES}, if the three points are on the same line. Otherwise, \textit{NO}.
\end{itemize}

Function \textit{In-Straight-Line-2} involves simple geometric calculations to check if the three input points are on the same straight line.

\textbf{Insight:} This function is called by robot $r_m$ with center $c_m$. The robot $r_m$ wants to check whether it is on the same line with its left nearest neighbor robot on the convex hull, $r_l$ with center $c_l$, and with its right nearest neighbor robot, $r_r$ with center $c_r$.

\section*{4 \textbf{Local Algorithm for Compute}}

In this section we present the algorithm that each robot runs locally while in state \textbf{compute}. This algorithm takes as an input the view of the robot (obtained in state \textbf{look}) and calculates the position in the plane the robot should move next (in state \textbf{move}).

In Section 4.1 we overview the states of the algorithm and in Section 4.2 we give a detail description of the algorithm along with key observations/properties.

\subsection*{4.1 States of the Algorithm}

Once a robot $r_i$ is in state \textbf{Compute} it starts executing the local algorithm $A_i$. Recall that $V_i$ denotes robot’s $r_i$ local view, that is, the set of robots that are visible to robot $r_i$ upon entering state \textbf{Compute}. The algorithm consists of 17 states. These states are algorithmic states within state \textbf{Compute} and we refer to them using the notation \textbf{Compute}.\langle \text{algorithm-state-name} \rangle. We now overview these states with respect to a robot $r_i$.

\begin{itemize}
\item 1. \textbf{Compute}.\textbf{Start}:
\begin{itemize}
\item This is the initial state of the algorithm run by robot $r_i$.
\end{itemize}
\item 2. \textbf{Compute}.\textbf{OnConvexHull}:
\begin{itemize}
\item Robot $r_i$ is on the convex hull formed by robots in its local view.
\end{itemize}
\item 3. \textbf{Compute}.\textbf{AllOnConvexHull}:
\begin{itemize}
\item Robot $r_i$ is on the convex hull.
\item Robot $r_i$ can see all other $n - 1$ robots (that is, it has full visibility).
\end{itemize}
\end{itemize}
- All other \( n - 1 \) robots are on the convex hull and have full visibility.

4. **Compute.Connected**:
   - Same conditions as in state 3
   - Robot \( r_i \) sees that all robots are connected.

5. **Compute.NotConnected**:
   - Same conditions as in state 3
   - Robot \( r_i \) sees that not all robots are connected.

6. **Compute.NotAllOnConvexHull**:
   - Robot \( r_i \) is on the convex hull.
   - Robot \( r_i \) cannot see all other \( n - 1 \) robots or at least one robot is not on the convex hull or all robots are on the convex hull, but there is at least one that does not have full visibility.

7. **Compute.NotOnStraightLine**:
   - Same conditions as in state 6
   - There are no two other robots on the same line with robot \( r_i \) (all three are on the convex hull).

8. **Compute.SpaceForMore**:
   - Same conditions as in state 7
   - Robot \( r_i \) sees that there is space on the convex hull for another robot. That is, there exist two neighboring robots on the convex hull that their distance is at least 2 (recall that robots are unit discs).

9. **Compute.NoSpaceForMore**:
   - Same conditions as in state 7
   - Robot \( r_i \) sees that there is no space on the convex hull for another robot; all neighboring robots on the convex hull have distance less than 2.

10. **Compute.OnStraightLine**:
    - Same conditions as in state 6
    - There are at least two other robots on the same line with robot \( r_i \) (all three are on the convex hull).

11. **Compute.SeeOneRobot**:
    - Same conditions as in state 10
    - Robot \( r_i \) can see only one robot on the line.

12. **Compute.SeeTwoRobot**:
    - Same conditions as in state 10
    - Robot \( r_i \) can see two robots on the line; this implies that robot \( r_i \) is between these two robots.
13. **Compute.NotOnConvexHull**:
   - Robot $r_i$ is enclosed in the convex hull formed by robots in its local view.

14. **Compute.IsTouching**:
   - Same conditions as in state 13.
   - Robot $r_i$ is touching another robot (the unit discs representing the robots are tangent).

15. **Compute.NotTouching**:
   - Same conditions as in state 13.
   - Robot $r_i$ does not touch any other robot.

16. **Compute.ToChange**:
   - Same conditions as in state 13.
   - If robot $r_i$ moves as calculated by the algorithm, then it will cause the convex hull to change, and this cannot be avoided.

17. **Compute.NotChange**:
   - Same conditions as in state 13.
   - If robot $r_i$ moves as calculated by the algorithm, then there is a way to avoid changing the convex hull.

Figure 4 depicts all possible states and transitions of the algorithm run by robot $r_i$. (For better readability the prefix *Compute* is voided.) The states that have no transition to another state are terminal, and they output the position that the robot will move next (and the robot exits state Compute and enters state Move). State **Compute.Connected** outputs the special point $\perp$ which causes robot $r_i$ to exit state Compute and enter state Terminate (the robot takes no further steps).

### 4.2 Description of the Algorithm

The algorithm consists of 17 procedures, each treating one of the possible algorithmic states. In particular, once the algorithm is in a state **Compute.(algorithm-state-name)** it runs the corresponding procedure algorithm-state-name that either implements a state transition or outputs a point on the plane the robot should move at (in the next state Move); it implements a state transition if it is in a non-terminal state and outputs a point otherwise. In a nutshell, the algorithm can be expressed as follows:

```plaintext
LOCAL ALGORITHM
if state= Compute.(algorithm-state-name) then run procedure algorithm-state-name.
```

We proceed to describe the procedures. The procedures are given with respect to a robot $r_i$.

#### 4.2.1 Procedure **Start**

Procedure: **Start**

**Precondition**: state= **Compute.Start**

**Effect**:
- **Call** function On-Convex-Hull with inputs, the local view $V_i$ and $c_i$, the center of robot $r_i$. 

---

15
Figure 4: All possible states and transitions of the algorithm run by robot $r_i$.

- If function $\text{On-Convex-Hull}$ returns YES then state := $\text{Compute.OnConvexHull}$
  Else state := $\text{Compute.NotOnConvexHull}$

Recall from Section 3.1 that function $\text{On-Convex-Hull}$ also returns the set of points that are on the convex hull based on robot’s $r_i$ view $V_i$. We will be denoting this set as $onCH(V_i)$. From this point onwards, robot $r_i$ carries the knowledge of $onCH(V_i)$ in the various algorithmic states, while in state $\text{Compute}$ (this knowledge is lost once it exists this state).

**Lemma 3** $\text{Start} (\langle \text{Compute.} \rangle) = \text{Compute.}(\langle \text{OnConvexHull} \rangle)$ iff $c_i \in onCH(V_i)$.

**Proof:** Procedure $\text{Start}$ uses function $\text{On-Convex-Hull}$, which uses Grahams algorithm\cite{12} that was proven to be correct.

### 4.2.2 Procedure $\text{OnConvexHull}$

**Procedure:** $\text{OnConvexHull}$

**Precondition:** state := $\text{Compute.OnConvexHull}$

**Effect:**
- If $|V_i| = n$ and $|onCH(V_i)| = n$ then
  - for each $r_j \in V_i - \{r_i\}$
    * Call function $\text{On-Straight-Line-2}$ with inputs: the center of the left neighbor of $r_j$, the center of $r_j$ and the center of the right neighbor of $r_j$.
    * If function $\text{On-Straight-Line-2}$ returned YES then state := $\text{Compute.NotAllOnConvexHull}$ and return
  - state := $\text{Compute.AllOnConvexHull}$
- Else state := $\text{Compute.NotAllOnConvexHull}$
Lemma 4  \( \text{OnConvexHull}(\text{Compute.}(\text{OnConvexHull})) = \text{Compute.}(\text{AllOnConvexHull}) \) iff \( |V_i| = n \) and \( |\text{onCH}(V_i)| = n \) and all robots have full visibility, according to \( V_i \).

Proof: Based on Lemma 3 \( c_i \in \text{onCH}(V_i) \). Then, there are four possible cases:

1. \( |V_i| < n \).
   In this case the procedure \( \text{OnConvexHull} \) will change state to \( \text{Compute.NotAllOnConvexHull} \) because if \( r_i \) can’t see all robots, it means that \( r_i \) does not have full visibility.

2. \( |V_i| = n \) and \( |\text{onCH}(V_i)| < n \).
   In this case procedure \( \text{OnConvexHull} \) will change state to \( \text{Compute.NotAllOnConvexHull} \) because if at least one robot is not on the convex hull, then the correct state to shift is \( \text{Compute.NotAllOnConvexHull} \). The procedure \( \text{OnConvexHull} \) can easily determine if all robots are on the convex hull by comparing the number of robots, with the number of robots that belong to \( \text{onCH}(V_i) \).

3. \( |V_i| = n \) and \( |\text{onCH}(V_i)| = n \) and at least one robot does not have full visibility.
   In this case procedure \( \text{OnConvexHull} \) will change state to \( \text{Compute.NotAllOnConvexHull} \) because if at least one robot does not have full visibility, the correct state to shift is \( \text{Compute.NotAllOnConvexHull} \). The procedure \( \text{OnConvexHull} \) can easily determine if all robots have full visibility (according to \( V_i \)) by checking if all robots are on the convex hull and no three robots are on the same line. From the definition of the convex hull it is clear that no incisions are allowed and hence if no three robots are on the same line, all robots will have full visibility.

4. \( |V_i| = n \) and \( |\text{onCH}(V_i)| = n \) and all robots have full visibility.
   In this case procedure \( \text{OnConvexHull} \) will change state to \( \text{Compute.AllOnConvexHull} \) because if all robots are on the convex hull and have full visibility, the correct state to shift is \( \text{Compute.AllOnConvexHull} \). The procedure \( \text{OnConvexHull} \) can easily determine if all robots have full visibility (according to \( V_i \)) by checking if all robots are on the convex hull and no three robots are on the same line. From the definition of the convex hull it is clear that no incisions are allowed and hence if no three robots are on the same line, all robots will have full visibility.

\[ \square \]

4.2.3 Procedure \text{AllOnConvexHull}

Procedure: \text{AllOnConvexHull}

Precondition: state= \text{Compute.AllOnConvexHull}

Effect:
- **Choose** a \( c_j \) from \( V_i \). Set \( \text{Component} = \{c_j\} \).
- **While** set \( \text{Component} \) changes do
  - For each \( c_j \in \text{Component} \) do
    * **Check** for each \( c_x \in V_i - \text{Component} - \{c_j\} \) whether its unit disc is tangent with the unit disc of \( c_j \). If true, then \( \text{Component} = \text{Component} \cup \{c_x\} \).
- **If** \( |\text{Component}| = n \) then state:= \text{Compute.Connected}
  Else state:= \text{Compute.NotConnected}

Lemma 5  \( \text{AllOnConvexHull}(\text{Compute.}(\text{AllOnConvexHull})) = \text{Compute.}(\text{Connected}) \) iff \( V_i \) is a connected configuration.

Proof: Based on Lemma 4 \( |\text{onCH}(V_i)| = n \) and all robots have full visibility. The procedure \( \text{AllOnConvexHull} \) uses simple geometric calculations to calculate the number of the robots that are connected to a random robot \( r_j \). If the number of the connected robots is \( n \), all robots are connected and therefore, the correct state to shift is \( \text{Compute.Connected} \).
4.2.4 Procedure Connected

Procedure: Connected

Precondition: state= Compute.Connected

Effect:
- Return the special point $\perp$.

Once this procedure is executed by robot $r_i$, it enters state terminate and does not perform any further steps.

4.2.5 Procedure NotConnected

High level idea: The purpose of this procedure is eventually all robots to form a connected configuration. This procedure gives priority to components with the smallest size firstly and secondly to components that the distance to their neighbor component on the right is the smallest distance between any two components. The rightmost robot of the component with the biggest priority moves to the left of its right neighbor component. If all components have equal priority (i.e. all components have the same size and the distance between any two components is the same) then robots start to converge. Robots can start moving only if for any three neighbor robots of the component, say $r_l, r_m$ and $r_r$ the vertical distance from line $r_l, r_r$ to $r_m$ is equal or more than $\frac{1}{n}$. We proceed with the complete code of the procedure.

Procedure: NotConnected

Precondition: state= Compute.NotConnected

Effect:
- Consider $c_m$, the center of the left neighbor of $r_i$ and $c_l$ the center of the left neighbor of $r_m$.
  - If the distance between the line $c_i,c_l$ and the point $c_m$ is $< \frac{1}{n}$, then Consider the cases that $r_i$ is in the middle or the right robot of three neighbor robots. Calculate $x$, the maximum distance that $r_i$ can move vertical to the line $c_i,c_l$ and with direction to the inside of the convex hull without causing any two other robots to be on a straight line or the distance between any line $c_l,c_r$ to $c_m$(any setting of three neighbor robots that involve $r_i$) be less than $\frac{1}{2n}$. $x'$ is the smallest between $\frac{1}{2n} - \epsilon$ and $x$. Consider the line that is vertical to the line $c_l,c_i$, starts from $c_i$ and has direction to the inside of the convex hull with an ending point $p$ with distance $x'$ from $c_l$. Return $p$.
  - If $r_i$ belongs to a set of continuous robots that the previous condition is true and the last robot on the right cannot move, then do exactly the same step as above but instead of the $r_i$ to be the right robot of three neighboring robots, to be the left with its right neighbors.

- If there exists a configuration of three neighbor robots $c_r, c_m$ and $c_l$ that the distance between the line $c_i,c_l$ and the point $c_m$ is $< \frac{1}{n}$, then Return $c_i$.

- If there is a $c_k \in V_i - \{c_l\}$ and a $c_j \in V_i - \{c_i\}$ that their unit disc is tangent with your unit disc, at your left and right respectively, then return $c_i$ ($r_i$’s current position).

- If All robots form one component and $r_i$ does not touch any other robot then Consider $c_l$ the center of the left neighbor of $r_i$, and $c_r$ the center of the right neighbor of $r_i$. Draw a line starting from $c_i$, vertical to line $c_l,c_r$ with direction to the inside of the convex hull and ending point $p$, $p$ is in distance $\frac{1}{2n}$ from $c_i$. Return $p$.

- Else call function In-Largest-Component with inputs, $V_i$ and $c_i$.

- If function In-Largest-Component returns 1, then return $c_i$. 

• **ElseIf** function `In-Largest-Component` returns 2, then **call** function `How-Much-Distance` with inputs, $V_i$ and $c_i$.

  - **If** function `How-Much-Distance` returns 1, **then call** function `Move-to-Point` with inputs, $c_i$ and $c_j$, where $c_j$ is $c_i$’s right neighbor on the Convex Hull. **Return** the point returned by `Move-to-Point`.

  - **ElseIf** function `How-Much-Distance` returns 2, **then call** function `Connected-Components` with inputs, $V_i$ and $c_i$. Let $(c_l, c_r)$ be the component that $c_i$ belongs $(c_l$ is the center of the left-most robot and $c_r$ the center of the right-most robot of the component). Draw a straight line between $c_l$ and $c_r$; call it $AB$. Draw a parallel line (wrt $AB$) such that it goes through $c_i$; call $c_i$, $C$ and the line $CD$. Draw a vertical line (wrt $CD$) from point $C$ towards the inside of Convex Hull with distance $\frac{1}{2n} - \epsilon$. If moving to $D$ does not cause robot $r_i$ to touch another unit disc in $r_i$’s component, or, if $c_i$ is $c_j$ or $c_r$, **then Return** $D$, else **return** $c_i$ (current position).

  - **Else Return** $c_i$.

• **ElseIf** function `In-Largest-Component` returns 3, then **call** function `In-Smallest-Component` with inputs, $V_i$ and $c_i$.

  - **If** function `In-Smallest-Component` returns 1, **then call** function `Move-to-Point` with inputs, $c_i$ and $c_j$, where $c_j$ is $c_i$’s right neighbor on the Convex Hull. **Return** the point returned by `Move-to-Point`.

  - **ElseIf** function `In-Smallest-Component` returns 2, **then call** function `How-Much-Distance` with inputs, $V_i$ and $c_i$.

    * **If** function `How-Much-Distance` returns 1, **then call** function `Move-to-Point` with inputs, $c_i$ and $c_j$, where $c_j$ is $c_i$’s right neighbor on the Convex Hull. **Return** the point returned by `Move-to-Point`.

    * **ElseIf** function `How-Much-Distance` returns 2, **then call** function `Connected-Components` with inputs, $V_i$ and $c_i$. Let $(c_l, c_r)$ be the component that $c_i$ belongs $(c_l$ is the center of the left-most robot and $c_r$ the center of the right-most robot of the component). Draw a straight line between $c_l$ and $c_r$; call it $AB$. Draw a parallel line (wrt $AB$) such that it goes through $c_i$; call $c_i$, $C$ and the line $CD$. Draw a vertical line (wrt $CD$) from point $C$ towards the inside of Convex Hull with distance $\frac{1}{2n} - \epsilon$. **Return** $D$.

    * **Else Return** $c_i$.

    - **Else Return** $c_i$.

**Lemma 6** The point returned by `NotConnected(Compute.(NotConnected))` keeps $V_i$ as a fully visible configuration and $|onCH(V_i)| = n$.

**Proof:** Based on Lemma 4 $|onCH(V_i)| = n$ and all robots have full visibility (that is, $V_i$ is a fully visible configuration). Based on Lemma 5, $V_i$ is not a connected configuration. We now show that in each of the following 6 possible cases, the point returned keeps $V_i$ as a fully visible configuration:

1. Robot $r_i$ touches one robot on its left and one robot on its right on the convex hull.

   In this case, procedure `NotConnected` will return $r_i$’s current position and hence it will not cause any change.

2. Robot $r_i$ is in the component with the most robots.

   In this case, procedure `NotConnected` will return $r_i$’s current position hence it will not cause any change.
3. All the components have the same number of robots and \( r_i \) is the rightmost robot of the component that has the smallest distance with its neighbor component, among any distance between any two adjacent components.

In this case, procedure NotConnected will call the function Move-To-Point, which will return a position on convex hull adjacent to the leftmost robot of the right neighbor component. Hence in this case robot \( r_i \)’s next position will be on the convex hull, \( r_i \) will have full visibility, will not be in the same line with any two other robots and will not cause any other robot not to be on the convex hull, provided that the other robots do not move.

4. All the components have the same number of robots and all the spaces between adjacent components are of the same distance.

In this case, procedure NotConnected will call the function Move-To-Point, which will return a position on convex hull adjacent to the leftmost robot of the right neighbor component. Hence in this case robot \( r_i \)’s next position will be on the convex hull, \( r_i \) will have full visibility, will not be in the same line with any two other robots and will not cause any other robot not to be on the convex hull, provided that the other robots do not move.

5. All the components have the same number of robots and \( r_i \) is not the rightmost robot of the component that has the smallest distance with its neighbor component, among any distance between any two adjacent components, and not all the spaces between adjacent components are of the same distance.

In this case, Procedure NotConnected will return \( r_i \)’s current position hence it will not cause any change.

6. Robot \( r_i \) is part of the component that has the smallest number of robots.

In this case, procedure NotConnected will call the function Move-To-Point, which will return a position on convex hull adjacent to the leftmost robot of the right neighbor component. Hence in this case robot \( r_i \)’s next position will be on the convex hull, \( r_i \) will have full visibility, will not be in the same line with any two other robots and will not cause any other robot not to be on the convex hull, provided that the other robots do not move.

4.2.6 Procedure NotAllOnConvexHull

Procedure: NotAllOnConvexHull

Precondition: state= Compute.NotAllOnConvexHull

Effect:
- Consider three points \( c_l, c_m, c_r \in onCH(V_i) \), where \( c_l \) is the left neighbor and \( c_r \) the right neighbor of \( c_m \), respectively, on the convex hull, and \( c_i \) is one of these points.
- Draw line segment \( \overline{c_l c_r} \). Let \( \overline{AB} \) be the line that is vertical to \( \overline{c_l c_r} \), it goes through \( c_l \) and both \( \overline{A c_l} \) and \( \overline{c_l B} \) have length \( \frac{1}{n} \). Line \( \overline{CD} \) is defined similarly for \( c_r \). (See Figure 5 for an example.)
- Consider all three cases, that is, \( c_i = c_l \), \( c_i = c_m \) or \( c_i = c_r \).
- If in any of the three cases, \( c_m \) is in the rectangle \( ABCD \), then state:= Compute.OnStraightLine

Else state:= Compute.NotOnStraightLine

Lemma 7 NotAllOnConvexHull(Compute.(NotAllOnConvexHull)) = Compute.(OnStraightLine) iff \( r_i \) is on the same line with any two other robots that are also on the convex hull.

Proof: Based on Lemma \( 3 \) robot \( r_i \in onCH(V_i) \). Based on Lemma \( 4 \) \( |V_i| \neq n \), or \( |onCH(V_i)| < n \) or not all robots have full visibility. Procedure NotAllOnConvexHull uses simple geometric calculations to determine if \( r_i \) is on straight line with its neighbor robots on \( onCH(V_i) \).
4.2.7 Procedure NotOnStraightLine

Procedure: NotOnStraightLine

Precondition: state = Compute.NotOnStraightLine

Effect:
- If $|onCH(V_i)| = n$ then state := Compute.SpaceForMore and Return
- If $|V_i| = n$ then
  - Check whether there exists a side on the Convex Hull with length of at least 2.
  - If there exists then state := Compute.SpaceForMore
    Else state := Compute.NoSpaceForMore
- Else
  - $\forall c_j \in onCH(V_i)$ Copy $c_j$ in a new set named $onCH2$
  - $\forall c_j \notin onCH(V_i)$ draw a straight line from $r_i$ that has an ending point $x$, $x \in onCH$ and $c_j$ is on that line.
  - Copy $x$ on $onCH2$
  - Check whether there exists a side on $onCH2$ with length of at least 2.
  - If there exists then state := Compute.SpaceForMore
    Else state := Compute.NoSpaceForMore

Lemma 8 NotOnStraightLine(Compute⟨NotOnStraightLine⟩) = Compute⟨SpaceForMore⟩ iff $|onCH(V_i)| = n$ or there exist a space, for at least one robot, between any two adjacent robots that are on the convex hull.

Proof: Based on Lemma 3 $r_i \in onCH$. Based on Lemma 4 $|V_i| \neq n$, or $|onCH(V_i)| < n$ or not all robots have full visibility. Based on Lemma 7 $r_i$ is not on a straight line with any two other robots that $\in onCH$. There are three possible cases:

1. $|onCH(V_i)| = n$. In this case, robot $r_i$ moves to the state Compute.SpaceForMore, because it is not necessary to create extra space on the convex hull for more robots, because all $n$ robots are already on convex hull.

2. $|onCH(V_i)| < n$ and there exist a space, for at least one robot, on the convex hull.

   In this case, procedure NotOnStraightLine uses simple calculations to determine if there exist a space for at least one robot on the convex hull. It calculates the distance between adjacent robots on the convex hull and if there exist at least two adjacent points that have more than 2 distance, the correct state to move is Compute.SpaceForMore.

3. $|onCH(V_i)| < n$ and no space, for at least one robot, on the convex hull exists.

   In this case procedure NotOnStraightLine uses simple calculations to determine if there exist a space for at least one robot on the convex hull. It calculates the distance between adjacent robots on the convex hull and if no such space exists, the correct state to move is Compute.SpaceForMore. ■
4.2.8 Procedure SpaceForMore

Procedure: SpaceForMore

Precondition: state= Compute.SpaceForMore

Effect:

• If \( r_i \) is tangent with a robot \( r_j, r_j \in onCH(V_i) \) that they are not adjacent on \( onCH(V_i) \) then consider \( c_l \) and \( c_r \), the centers of \( r_i \)'s left and right neighbor on the convex hull respectively. Draw a straight line starting from \( c_i \), vertical to the line \( c_l,c_r \), with direction outside of the convex hull and at distance \( \frac{1}{2n} - \epsilon \). The ending point of this line is \( p \). Return \( p \).

• Else Return \( c_i \)

The reason that \( p \) is outside of the convex hull by a distance \( \frac{1}{2n} - \epsilon \) is because if two robots are not adjacent on the convex hull and are touching, it means that it is possible to obstruct other robots from seeing each other.

Lemma 9 SpaceForMore(Compute.(SpaceForMore)) = \( c_i \) iff \( r_i \) is not tangent with any robot \( r_j, r_j \in onCH(V_i) \) that they are not adjacent on \( onCH(V_i) \), else SpaceForMore(Compute.(SpaceForMore)) = \( p \) were \( p \) is at distance \( \frac{1}{2n} - \epsilon \) outside of \( onCH(V_i) \).

Proof: Based on Lemma 3 \( r_i \in onCH(V_i) \). Based on Lemma 4 \( |V_i| \neq n \), or \( |onCH(V_i)| < n \) or not all robots have full visibility. Based on Lemma 7 \( r_i \) is not on a straight line with any two other robots that \( \in onCH(V_i) \). Based on Lemma 8 there exist a space for at least one robot on \( onCH(V_i) \). There are are two cases:

If \( r_i \) is not tangent with any robot \( r_j \) such that \( r_j \in onCH(V_i) \) and \( r_j \) is not adjacent to \( r_i \) on \( onCH(V_i) \), then procedure SpaceForMore returns \( c_i \), else it returns a point \( p \) at distance \( \frac{1}{2n} - \epsilon \) outside of \( onCH(V_i) \).

4.2.9 Procedure NoSpaceForMore

Procedure: NoSpaceForMore

Precondition: state= Compute.NoSpaceForMore

Effect:

• Let \( c_l \) be the center of \( c_i \)'s left neighbor and let \( c_r \) be the center of \( c_i \)'s right neighbor on the Convex Hull. Draw a straight line between \( c_l \) and \( c_r \); call it \( AB \).

• Let \( m \) be the center of line \( AB \).

• Draw a vertical line (wrt \( AB \)) starting from \( m \) and ending at distance \( \frac{1}{2n} - \epsilon \) away from the Convex Hull; call the ending point, \( p \).

• Calculate the point on the line between \( m \) and \( p \), which \( r_i \) can move to maximum distance from \( m \) without causing \( onCH(V_i) \) to change; call this point \( p' \).

• Return \( p' \).

Lemma 10 NoSpaceForMore(Compute.(NoSpaceForMore)) = \( p \) were \( p \) is at distance \( \frac{1}{2n} - \epsilon \) outside of \( onCH(V_i) \).

Proof: Based on Lemma 3 \( r_i \in onCH \). Based on Lemma 4 \( |V_i| \neq n \), or \( |onCH(V_i)| < n \) or not all robots have full visibility. Based on Lemma 7 \( r_i \) is not on a straight line with any two other robots that \( \in onCH \). Based on Lemma 8 no space exists for at least one robot on \( onCH(V_i) \).

The correctness of the Lemma follows from the code of the procedure.
4.2.10 Procedure OnStraightLine

Procedure: OnStraightLine

Precondition: state= Compute.OnStraightLine

Effect:
- Consider the same setting as in procedure NotAllOnConvexHull.
- If for one of the cases \( c_m \) is in the rectangle \( ABCD \) and holds that \( c_m = c_i \), then state:= Compute.SeeTwoRobot
  Else state:= Compute.SeeOneRobot

Lemma 11 OnStraightLine\( (\text{Compute.} \langle \text{OnStraightLine} \rangle) = \text{Compute.} \langle \text{SeeTwoRobots} \rangle \) iff \( r_i \) is on the same line with two robots on the convex hull, its left neighbor \( r_l \), and its right neighbor \( r_r \).

Proof: Based on Lemma \( 3 \) robot \( r_i \in \text{onCH}(V_i) \). Based on Lemma \( 4 \) \( |V_i| \neq n \), or \( |\text{onCH}(V_i)| < n \) or not all robots have full visibility. Based on Lemma \( 7 \) \( r_i \) is on straight line with its neighbor robots on the convex hull. Procedure OnStraightLine uses simple geometric calculations to determine if robot \( r_i \) is in the middle of \( c_l \) and \( c_r \). ■

4.2.11 Procedure SeeOneRobot

Procedure: SeeOneRobot

Precondition: state= Compute.SeeOneRobot

Effect:
- Return \( c_i \) (current position).

Then, trivially:

Lemma 12 SeeOneRobot\( (\text{Compute.} \langle \text{SeeOneRobot} \rangle) = c_i \).

4.2.12 Procedure SeeTwoRobot

Procedure: SeeTwoRobot

Precondition: state= Compute.SeeTwoRobot

Effect:
- Consider the same setting as in procedure NotAllOnConvexHull and \( c_m = c_i \).
- Consider the line segment that is vertical to line \( \overline{c_l c_r} \), it starts from \( c_i \) with direction outside of the convex hull (if this is not possible to determine choose a random direction) and ends at distance \( \frac{1}{2n} - \epsilon \) from \( c_i \). Call the ending point, \( p \). Consider the line segment that is vertical to line \( \overline{c_l c_r} \), it starts from line \( \overline{c_l c_r} \) and ends at distance \( \frac{1}{n} \) from line \( \overline{c_l c_r} \). Call the ending point, \( p' \), such that \( c_i \) is in the same line with and between \( p' \) and line \( \overline{c_l c_r} \).
- Return the point that is nearest to \( c_i \), between \( p \) and \( p' \).

Lemma 13 The point \( p \) returned by SeeTwoRobot\( (\text{Compute.} \langle \text{SeeTwoRobots} \rangle) \) is such that if robot \( r_i \) moves there (\( c_i \) is on \( p \)), then \( r_i \) will no longer be in a straight line with it’s two adjacent robots on the convex hull, provided that the other robots do not move.

Proof: Based on Lemma \( 3 \) robot \( r_i \in \text{onCH}(V_i) \). Based on Lemma \( 4 \) \( |V_i| \neq n \), or \( |\text{onCH}(V_i)| < n \) or not all robots have full visibility. Based on Lemma \( 7 \) \( r_i \) is on straight line with its neighbor robots on the convex hull. Based on Lemma \( 11 \) \( r_i \) is on the same line and in the middle of its left neighbor robot on the convex hull \( r_l \) and with its right neighbor robot on the convex hull \( r_r \). Procedure SeeTwoRobot results robot \( r_i \) to move at distance \( \frac{1}{2n} - \epsilon \) from its current position, with direction out of the convex hull as described in the Procedure SeeTwoRobot. After moving in this position robot \( r_i \) will no longer be on a straight line with its adjacent robots on the convex hull. ■
**Figure 5:** An example where the unit disc (robot) with center $c_m$ intersects rectangle $ABCD$.

### 4.2.13 Procedure NotOnConvexHull

**Procedure:** NotOnConvexHull  
**Precondition:** state = Compute.NotOnConvexHull  
**Effect:**
- Check whether there is a $c_j \in V_i$ so that the unit disc with center $c_j$ is tangent with the unit disc with center $c_i$.  
- If yes, then state := Compute.IsTouching  
  Else state := Compute.NotTouching

**Lemma 14** NotOnConvexHull(Compute.(NotOnConvexHull)) = Compute.(IsTouching) iff $r_i$’s unit disk is tangent with a unit disk of any other robot.

**Proof:** Based on Lemma 3, robot $r_i /\notin onCH(V_i)$. Now the claim of the lemma follows by close investigation of procedure NotOnConvexHull.

### 4.2.14 Procedure IsTouching

**Procedure:** IsTouching  
**Precondition:** state = Compute.IsTouching  
**Effect:**
- Call function Find-Points with input $onCH(V_i)$.  
- If Function Find-Points returned one or more points then choose $p$, the point returned by Function Find-Points that is closest to $c_i$  
  - If one of the robots that are touching $r_i$ is closer to $p$ than $r_i$ then return $c_i$ (current position).  
  - Else If one or more of the robots that are touching $r_i$ have the same distance with $r_i$ to $p$ then  
    - If $r_i$ is the rightmost of the robots that are touching and have the same distance to $p$ then $p' \in onCH(V_i)$ and $p'$ is on the line between $c_i$ and $p$. Return $p'$.  
    - Else return $c_i$ (current position).  
  - Else $p' \in onCH(V_i)$ and $p'$ is on the line between $c_i$ and $p$. Return $p'$.  
- Else choose the two closest neighboring robots, that have distance of at least 2, to $c_i$ that are on the Convex Hull.  
  - If No neighboring robots on the convex hull have distance of at least 2 then return $c_i$ (current position).  
  - If one of the robots that are touching $r_i$ is closer to these robots than $r_i$ then return $c_i$ (current position).  
  - Else If one or more of the robots that are touching $r_i$ have the same distance with $r_i$ to these robots then
If \( r_i \) is the rightmost of the robots that are touching and have the same distance to the closest robots on the convex hull then draw a straight line between the centers of the two closest robots to \( r_i \) on the convex hull and find the center of this line, \( p \). Return \( p \).

Else return \( c_i \) (current position).

– Else draw a straight line between the centers of these robots and find the center of this line, \( p \). Return \( p \).

In a given set of robots, we consider that a robot has higher proximity compared to the other robots of that set if it is the closest to its closest space on the convex hull or to the closest space that Function \text{FindPoints} \text{returned} (depending on the case). If more than one robots of that set of robots have the same distance to the closest space, then the rightmost of these robots has the highest proximity (straight direction is considered to be to the outside of the convex hull of the target point).

**Lemma 15** \( \text{IsTouching} \langle \text{Compute.} \rangle \) will result robot \( r_i \)'s unit disk to no longer be tangent with any other robot's unit disk (from the robots that \( r_i \) touches) if \( r_i \) has the highest proximity (from the robots that are touching). If no space of at least 2 exists on the convex hull, then \( r_i \) stays in the same position.

**Proof:** Based on Lemma 3 \( r_i \notin onCH(V_i) \). Based on Lemma 4 robot \( r_i \)'s unit disk is tangent with at least one other robot's unit disk.

There are 9 possible cases:

1. Function \text{FindPoints} returned one point or more and \( r_i \) is not the closest robot (from the robots that are touching) to \( p \).
   In this case, \( r_i \) will remain in the same position because it does not have the highest proximity.

2. Function \text{FindPoints} returned one point or more. \( r_i \) is the closest robot (from the robots that are touching) to \( p \) and has the same distance with \( p \) with at least another robot. \( r_i \) is the rightmost robot of the robots that are touching and have the same distance to \( p \).
   In this case, \( r_i \) will move to \( p' \) because it has the highest proximity.

3. Function \text{FindPoints} returned one point or more. \( r_i \) is the closest robot (from the robots that are touching) to \( p \) and has the same distance with \( p \) with at least another robot. \( r_i \) is not the rightmost robot of the robots that are touching and have the same distance to \( p \).
   In this case, \( r_i \) will remain in the same position because it does not have the highest proximity.

4. Function \text{FindPoints} returned one point or more. \( r_i \) is the closest robot (from the robots that are touching) to \( p \) and no other robot (from the robots that are touching) has the same distance to \( p \) with \( r_i \).
   In this case, \( r_i \) will move to \( p' \) because it has the highest proximity.

5. Function \text{FindPoints} did not returned any point. No space of at least 2 exists on the convex hull.
   In this case, \( r_i \) will remain in the same position because not enough space for it exists on the convex hull.

6. Function \text{FindPoints} did not returned any point. \( r_i \) is not the closest robot (from the robots that are touching) to the two closest robots (from \( r_i \)) on the convex hull.
   In this case, \( r_i \) will remain in the same position because it does not have the highest proximity.
7. Function `FindPoints` did not returned any point. \( r_i \) is the closest robot (from the robots that are touching) to the two closest robots (from \( r_i \)) on the convex hull and has the same distance with at least another robot. \( r_i \) is the rightmost robot of the robots that are touching and have the same distance to the two closest robots (from \( r_i \)) on the convex hull.

In this case, \( r_i \) will move to the center of the line between those two robots because it has the highest proximity.

8. Function `FindPoints` did not returned any point. \( r_i \) is the closest robot (from the robots that are touching) to the two closest robots (from \( r_i \)) on the convex hull and has the same distance with \( p \) with at least another robot. \( r_i \) is not the rightmost robot of the robots that are touching and have the same distance to the two closest robots (from \( r_i \)) on the convex hull.

In this case, \( r_i \) will remain in the same position because it does not have the highest proximity.

9. Function `FindPoints` did not returned any point. \( r_i \) is the closest robot (from the robots that are touching) to the two closest robots on the convex hull and no other robot (from the robots that are touching) has the same distance to those robots with \( r_i \).

In this case, \( r_i \) will move to the center of the line between those two robots because it has the highest proximity.

\[ \text{Lemma 16} \ IsTouching(\text{Compute.}(IsTouching)) \text{ will result at least one of the robots that are touching to move with direction to the convex hull, if a space of at least 2 exists on the convex hull.} \]

**Proof:** The claim follows from the code of the procedure.

\[ \text{4.2.15 Procedure NotTouching} \]

**Procedure:** NotTouching

**Precondition:** state= Compute.NotTouching

**Effect:**
- Call function `Find-Points` with input \( onCH(V_i) \).
- If function `Find-Points` returns at least one point, then state:= Compute.NotChange
  Else state:= Compute.ToChange

\[ \text{Lemma 17 NotTouching(Compute.(NotTouching))} = \text{Compute.(NotChange)} \text{ iff } r_i \text{ can move on the convex hull, without causing any additional change on the convex hull.} \]

**Proof:** Based on Lemma 3 robot \( r_i \notin onCH(V_i) \). Based on Lemma 14 robot \( r_i \) is not touching with any other robot. Procedure NotTouching calls Function `Find-Points`. Per Lemma 1 function `Find-Points` returns all the possible points on convex hull, that \( r_i \) can move to, without causing any changes to \( onCH(V_i) \), provided that other robots do not move. If Function `Find Points` does not return any points, the next state will correctly be Compute.ToChange. Otherwise, the correct state to move is Compute.NotChange.
4.2.16 Procedure ToChange

Procedure: ToChange

Precondition: state= Compute.ToChange

Effect:
- Among the robots that are neighboring on the convex hull and have distance at least 2 from each-other, choose the two closest ones to \( r_i \).
- If no neighbor robots have distance greater or equal with 2, Return \( c_i \)
- Else Draw a straight line between the centers of these robots and find the center of this line, \( p \). Return \( p \).

Lemma 18 ToChange(Compute.(ToChange)) = \( p \), when \( p \in onCH(V_i) \) if there exists a space of at least 2 on the convex hull. Else ToChange(Compute.(ToChange)) = \( c_i \)

Proof: Based on Lemma 3 \( r_i \notin onCH(V_i) \). Based on Lemma 14 \( r_i \) is not tangent with any other robot. Based on Lemma 17 \( r_i \) cannot move to the convex hull without causing it to change. The claim now follows from the code of the procedure.

4.2.17 Procedure NotChange

Procedure: NotChange

Precondition: state= Compute.NotChange

Effect:
- Call function Find-Points with input onCH(Vi).
- Choose the point returned by function Find-Points that is closest to \( c_i \); call it \( x \).
- \( p \in onCH \) and \( p \) is in the line between the points \( c_i \) and \( x \).
- Return \( p \).

Lemma 19 NotChange(Compute.(NotChange)) = \( p \), where \( p \in onCH(V_i) \).

Proof: Based on Lemma 3 \( r_i \notin onCH(V_i) \). Based on Lemma 14 \( r_i \) is not tangent with any other robot. Based on Lemma 17 \( r_i \) can move to the convex hull without causing it to change. Then it follows that the returned point is on the convex hull.

5 Distributed Algorithm for Gathering

The high level idea of the algorithm is as follows: The objective is for the robots to form a convex hull and be able to see each other. Once this is achieved, then the robots start to converge (they get closer), while maintaining the convex hull formation, so that they form a connected component. It follows that when all robots are on the convex hull, they can see each other, and are connected, the gathering problem is solved and each robot terminates.

The distributed algorithm is essentially composed of the asynchronous execution of the robots’ state transition cycle (including the local algorithm when in state Compute).

We now proceed to show that the distributed algorithm correctly solves the gathering problem. We first provide some important definitions and then we proceed with the proof of correctness. Robots make decisions based on their local views, but due to asynchrony, each robot’s local view might not reflect the current system configuration. Hence, our proof shows that the local decisions made by the robots are designed in such a way, that robots can coordinate correctly in the face of asynchrony and hence reach a solution to the gathering problem.
5.1 Definitions

Given a Robot Configuration $\mathcal{R}$, we denote by $\mathcal{G}_\mathcal{R}$ the geometric configuration and by $\mathcal{S}_\mathcal{R}$ the state configuration of $\mathcal{R}$. Recall that for a geometric configuration $\mathcal{G}$, we denote by $CH(\mathcal{G})$ the convex hull formed by the points in $\mathcal{G}$, as output by Graham’s Algorithm. Also, we denote by $onCH(\mathcal{G}) \subseteq \mathcal{G}$ the set of points in $\mathcal{G}$ that are on the convex hull.

**Bad Configurations.** We say that a robot configuration $\mathcal{R}_x$ is a bad configuration, when one of the two following cases is true:

1. Bad configuration of Type 1. When all of the following hold:
   - Configuration $\mathcal{G}_{\mathcal{R}_x}$ is fully visible and $|onCH(\mathcal{G}_{\mathcal{R}_x})| = n$;
   - A robot $r_i$ in this configuration has as local view $V_i$ a previous configuration $\mathcal{G}_{\mathcal{R}_y}$, $y < x$, such that $|onCH(\mathcal{G}_{\mathcal{R}_y})| < n$, $r_i \in onCH(\mathcal{R}_y)$ and $r_i$ sees that no space for more robots to get on the convex hull exists.

2. Bad configuration of Type 2. When all of the following hold:
   - Configuration $\mathcal{G}_{\mathcal{R}_x}$ is fully visible and $|onCH(\mathcal{G}_{\mathcal{R}_x})| = n$;
   - There exists a preceding configuration $\mathcal{G}_{\mathcal{R}_y}$, $y < x$, in which at least four robots, call them $r_1, r_{m1}, r_{m2}$ and $r_r$, are on a straight line and $r_1, r_{m1}, r_{m2}, r_r \in onCH(\mathcal{G}_{\mathcal{R}_y})$.

Both types are considered bad, because they can potentially lead to a succeeding configuration (wrt $\mathcal{R}_x$) that is no longer fully visible or all robots are on the convex hull; a property that we would like, once it holds, to continue holding for all succeeding configurations.

Let us explain how this is possible, first for the bad configuration of type 1. According to the local algorithm, when robot $r_i$ witness a view as described in the second bullet of type 1 configuration, robot $r_i$ must start moving with direction outside of the convex hull so to make space for more robots to get on the convex hull. This is also the case for all robots sharing the same or similar view with $r_i$. When $r_i$ starts moving (it gets in state move), the adversary can impose the following strategy: It makes $r_i$ to “move too slow” and lets the other robots move with such “a speed” that the robots reach configuration $\mathcal{R}_x$. Since $r_i$ has not changed its state (it is still in state move), it continues to move outside of the convex hull. This may cause a neighboring robot of $r_i$ not to be on the convex hull anymore or not be able to see all robots. Hence, while $\mathcal{G}_{\mathcal{R}_x}$ was a fully visible configuration and $|onCH(\mathcal{G}_{\mathcal{R}_x})| = n$, it is possible for a succeeding configuration not to have one (or both) of the these properties anymore.

Now we consider a type 2 bad configuration. According to the local algorithm, if robots $r_1, r_{m1}, r_{m2}, r_r$ witness configuration $\mathcal{G}_{\mathcal{R}_y}$, then robots $r_{m1}$ and $r_{m2}$ must start moving with direction outside of the convex hull (the robots that realize they are in the middle of the straight line must move outside so to enable the “edge” robots to see each other; the “edge” robots do not move). When $r_{m1}$ and $r_{m2}$ start moving (they get in state move) the adversary can impose the following strategy: It lets robot $r_{m1}$ to move slightly and then it stops it (with a stop($r_{m1}$) event). It lets robot $r_{m2}$ to move slightly and then the adversary makes it to move very slow (so robot $r_{m2}$ is still in state move). The adversary could stop robot $r_{m1}$ and delay $r_{m2}$ in such a way that configuration $\mathcal{R}_x$ is reached (recall that $|onCH(\mathcal{G}_{\mathcal{R}_x})| = n$ and $\mathcal{G}_{\mathcal{R}_x}$ is a fully visible configuration). But since $r_{m2}$ continues to move, it is possible to cause robot $r_{m1}$ to no longer be in onCH or some other robot (including $r_{m2}$) not be able to see all other robots. Hence it is possible for a succeeding configuration of $\mathcal{G}_{\mathcal{R}_x}$ not to have one (or both) of the these properties anymore.

28
Safe Configurations. We say that a robot configuration $\mathcal{R}$ is a safe configuration, when the following is true:

$$|onCH(\mathcal{G}_R)| = n, \mathcal{G}_R \text{ is a fully visible configuration and } \forall r_i, |onCH(V_i)| = n \text{ and } V_i \text{ is a fully visible configuration (that is, all robots know that the configuration is fully visible).}$$

The reason we consider these configurations as safe, is because, as we will show later, once an execution of the algorithm reaches such a safe configuration, then no succeeding configuration can be a bad configuration.

We define a bad execution fragment (resp. execution) of the algorithm to be an execution fragment (resp. execution) that contains at least one bad robot configuration. Similarly, we define a good execution fragment (resp. execution) to be an execution fragment (resp. execution) that contains only good configurations.

5.2 Proof of Correctness

The proof is broken into two parts. In the first part we prove safety and liveness properties considering only good execution fragments. Then we show that the algorithm is correct for any execution (including ones containing bad configurations).

5.2.1 Good Executions

In the section (with the exception of the first lemma) we consider only executions and executions fragments that are good, that is, they do not consider bad configurations. We first prove safety and then liveness properties for such executions.

Safety Properties

The following lemma states that as long as not all robots are on the convex hull, or even if all robots are on the convex hull but there is at least one robot that cannot see all other robots, then the convex hull can only expand. (This property holds even for bad execution fragments).

**Lemma 20** Given an execution fragment $\mathcal{R}_0, e_1, ..., \mathcal{R}_{m-1}$ such that for all $\mathcal{R}_k$, $0 \leq k \leq m - 1$ holds that:

- $c1$: $|onCH(\mathcal{G}_{R_k})| < n$ or $c2$: $|onCH(\mathcal{G}_{R_k})| = n$ and $\mathcal{G}_{R_k}$ is not a fully visible configuration, then for any step $(\mathcal{R}_{m-1}, e_m, \mathcal{R}_m)$, $CH(\mathcal{G}_{R_{m-1}}) \subseteq CH(\mathcal{G}_{R_m})$

**Proof:** The possible events $e_m$ are:

(A) $e_m$ involves (directly) robot $r_i$. If $r_i$ in $\mathcal{R}_{m-1}$ is in state **Wait**, **Look** or **Compute**, then it trivially holds that none of the possible events $e_m$ can affect the CH. So, it remains to consider the case that $r_i$ in $\mathcal{R}_{m-1}$ is in state **Move**. In this case, there are three possible cases for event $e_m$: stop($r_i$), arrive($r_i$) or collide($X$), $r_i \in X$.

Since $r_i$ is in state **Move**, then it is following a trajectory (start,target), where start is the position of its center when it start moving, and target is the position it wants to reach, as it was calculated when the robot was in state **Compute** (it is possible that start = target), say in $\mathcal{R}_k$, $k < m - 1$. Furthermore, $r_i$ made decisions based on the view the robot obtained while in state **Look**, in some configuration $\mathcal{R}_{k'}$, $k' < k$. It follows that $k' < m - 1$, hence the lemma Hypothesis applies (i.e., for $\mathcal{G}_{R_{k'}}$, either property $c1$ or $c2$ hold). In other words, for configurations $R_{k'}$ through $R_{m-1}$, $V_i \subseteq \mathcal{G}_{R_{k'}}$. Now, for $V_i$ we have the following possible cases:
• \( r_i \in onCH(V_i) \). Based on Lemma 3 and Function Start, \( r_i \) first gets into state Compute.OnConvexHull. Then, based on Lemma 4 and Function OnConvexHull, \( r_i \) gets into state Compute.NotAllOnConvexHull, since \( c_1 \) or \( c_2 \) is true for \( G_{R_k} \). Now the following are possible:

- \( r_i \) is on straight line with two other robots that are also on the convex hull. In this case, per Lemma 7 and Function NotAllOnConvexHull, robot \( r_i \) gets into state Compute.OnStraightLine. Then two cases are possible:
  * \( r_i \) is in the middle of the two other robots. Based on Lemma 11 and Function OnStraightLine, robot \( r_i \) gets to state Compute.SeeTwoRobots and runs Procedure SeeTwoRobots. Based on Procedure SeeTwoRobots and per Lemma 13, the procedure returns a point \( p \) with direction away from the convex hull (as witnessed in view \( V_i \) in configuration \( R_{k'} \)). If \( e_m \) is \( Stop(r_i) \) or \( Collide(X) \), \( r_i \in X \), \( r_i \)'s position in \( R_m \) is a point between the trajectory \( (c_i, p) \), \( c_i \) being the position of \( r_i \)'s in \( V_i \). Hence \( CH(G_{R_{m-1}}) \) can only increase (it certainly cannot decrease since it is moving out of the convex hull). If \( e_m = Arrive(r_i) \), then \( r_i \) reaches point \( p \), which again means that \( CH(G_{R_{m-1}}) \) can only increase.
  * \( r_i \) is not in the middle of the two other robots. Based on Lemma 11 and Function OnStraightLine, robot \( r_i \) gets into state Compute.SeeOneRobot and runs Procedure SeeOneRobot. Based on Lemma 12 the procedure returns \( c_i \), that is, the robot does not move. Hence \( r_i \) does not cause \( CH(G_{R_{m-1}}) \) to change.

- \( r_i \) is not on a straight line with any two other robots that are also on the convex hull. Per Lemma 7 and Function NotAllOnConvexHull, robot \( r_i \) gets into state Compute.NotOnStraightLine. Then the following cases are possible:
  * Condition \( c_1 \) holds and \( r_i \) sees that there exist enough space for at least one robot to get on the convex hull. Then, per Lemma 8 and Function NotOnStraightLine, robot \( r_i \) gets into state Compute.SpaceForMore and runs Procedure SpaceForMore, per Lemma 9 it returns \( c_i \), that is, the robot does not move, or it moves with direction outside of the convex hull. Hence \( r_i \) does not cause \( CH(G_{R_{m-1}}) \) to change or it causes \( CH(G_{R_{m-1}}) \) to increase.
  * Condition \( c_1 \) holds and \( r_i \) sees that there is not enough space for at least one robot to get on the convex hull. In this case, per Lemma 8 and Function NotOnStraightLine, robot \( r_i \) gets into state Compute.NoSpaceForMore and runs Procedure NoSpaceForMore. Based on Lemma 10 the procedure returns a point \( p \) with direction away from the convex hull (as witnessed in view \( V_i \) in configuration \( R_{k'} \)). Then, using the exact reasoning as above (when the Procedure SeeTwoRobots is run), it follows that \( CH(G_{R_{m-1}}) \) can only increase.
  * The case that Condition \( c_2 \) holds is handled identically as above, depending what \( r_i \) sees.

• \( r_i \notin onCH(V_i) \). (Only when condition \( c_1 \) holds.) Based on Lemma 3 and Function Start, \( r_i \) gets into state Compute.NotOnConvexHull. Then we have the following cases.

  - \( r_i \) is touching another robot. Based on Lemma 14 and Function NotOnConvexHull, \( r_i \) gets into state Compute.IsTouching and runs Procedure IsTouching, based on Lemma 15 it returns a point \( p \in onCH(G_{R_{k'}}) \) (that is, a point towards the witnessed convex hull) or \( c_i \). So, this means, regardless if \( e_m \) is a \( Stop, Collide \) or \( Arrive \) event on \( r_i \), robot \( r_i \) can reach up to the boundary of \( CH(G_{R_{k'}}) \). Then it is not difficult to see that \( r_i \) does not change \( CH(G_{R_{m-1}}) \) to change (\( r_i \) will either be on the boundary or inside of \( CH(G_{R_{m-1}}) \)).
Lemma 21

Given a good execution fragment $\mathcal{R}_x, e_x, \ldots, \mathcal{R}_{m-1}$ such that $\forall k, x \leq k \leq m-1$ holds that

1. $|\text{onCH}(\mathcal{G}_{\mathcal{R}_k})| = n$ and $\mathcal{G}_{\mathcal{R}_k}$ is a fully visible configuration

AND

2. $\mathcal{G}_{\mathcal{R}_k}$ is not a connected configuration,

then for any step $(\mathcal{R}_{m-1}, e_m, \mathcal{R}_m)$, c1 holds for $\mathcal{G}_{\mathcal{R}_m}$ and $CH(\mathcal{G}_{\mathcal{R}_{m-1}}) \supseteq CH(\mathcal{G}_{\mathcal{R}_m})$

Proof: The possible events $e_m$ are:

1. $e_m$ involves (directly) robot $r_i$. If $r_i$ in $\mathcal{R}_{m-1}$ is in state Wait, Look or Compute, then it trivially holds that none of the possible events $e_m$ can affect the $CH$. So, it remains to consider the case that $r_i$ in $\mathcal{R}_{m-1}$ is in state Move. In this case, there are three possible cases for event $e_m$: stop($r_i$), arrive($r_i$) or collide($X$), $r_i \in X$.

(A) Since $r_i$ is in state Move, then it is following a trajectory $(\text{start}, \text{target})$, where $\text{start}$ is the position of its center when it start moving, and $\text{target}$ is the position it wants to reach, as it was calculated when the robot was in state Compute (it is possible that $\text{start} = \text{target}$), say in $\mathcal{R}_k$, $k < m - 1$. Furthermore, $r_i$ made decisions based on the view the robot obtained while in state Look, in some configuration $\mathcal{R}_{k'}$, $k' < k$. It follows that $k' < m - 1$, hence the lemma Hypothesis applies (i.e., for $\mathcal{G}_{\mathcal{R}_{k'}}$, properties c1 and c2 hold). In other words, for configurations $\mathcal{R}_{k'}$ through $\mathcal{R}_{m-1}$, $V_i \subseteq \mathcal{G}_{\mathcal{R}_{k'}}$. Now, for $V_i$ we have the following possible cases:

- Robot $r_i \in \text{onCH}(\mathcal{G}_{\mathcal{R}_{k'}})$, $|\text{onCH}(\mathcal{G}_{\mathcal{R}_{k'}})| = n$ and $\mathcal{G}_{\mathcal{R}_{k'}}$ is a fully visible configuration because c1 is true.
Based on Lemma 3 and Function Start, \( r_i \) moves to state **OnConvexHull**. Based on Lemma 4 and Function OnConvexHull, \( r_i \) moves to state **AllOnConvexHull**. Based on Lemma 5 and Function AllOnConvexHull, \( r_i \) moves to state **NotConnected**. Procedure NotConnected returns a point \( p \in CH(\mathcal{G}_{k_i}) \). Three possible events can happen:

- **Stop\((r_i)\)** or **Collide\((r_i)\)**
  \( r_i \) moves a distance of at least \( \delta \) with direction from \( c_i \) to \( p \). Because of Lemma 6, \( r_i \) does not cause \(|onCH(\mathcal{G}_{k_i})| < n \) or \( \mathcal{G}_{k_i'} \) to be not a fully visible configuration. Because \( p \in CH(\mathcal{G}_{k_i}) \) and \( p \notin onCH(\mathcal{G}_{k_i'}) \), it follows that \( CH(\mathcal{G}_{k_i}) \) can only shrink.

- **Arrive\((r_i)\)**
  \( r_i \) moves to \( p \). Because of Lemma 6, \( r_i \) does not cause \(|onCH(\mathcal{G}_{k_i})| < n \) or \( \mathcal{G}_{k_i'} \) to be not a fully visible configuration. Because \( p \in CH(\mathcal{G}_{k_i}) \) and \( p \notin onCH(\mathcal{G}_{k_i'}) \), it follows that \( CH(\mathcal{G}_{k_i}) \) can only shrink.

- **Otherwise**
  This case is not possible, since \( c_1 \) is true.

Another robot \( r_j \) could also be in state **Move** in \( e_m \). We get the following cases:

a) \( r_j \) is in a trajectory *(start,target)*, that was decided on a robot configuration, say \( \mathcal{R}_k \). It follows that \( x < k < m - 1 \), hence Lemma Hypothesis applies. Specifically, \( r_j \) had a view where \( c_1 \) and \( c_2 \) were true for \( \mathcal{R}_k \). This is the same case with \( r_i \) (previous).

b) \( r_j \) is in a trajectory *(start,target)*, that was decided on a robot configuration, say \( \mathcal{R}_k \). It follows that \( k < x \).

- **\( c_1 \) and \( c_2 \) in \( \mathcal{R}_k \) were true**
  This is the same case with 1-A

- **\( c_1 \) was not true in \( \mathcal{R}_k \)**. We get the following cases:

  - **\( r_j \in onCH(\mathcal{G}_{k_i}) \)**
    Based on Lemma 3 and Function Start, \( r_j \) moves to state **Compute.OnConvexHull**. Based on Lemma 4 and Function OnConvexHull, \( r_j \) changes to state **Compute.NotAllOnConvexHull**, because \(|onCH(\mathcal{G}_{k_i})| < n \).

    - **\( r_j \) is on straight line with any two other robots that \( \in onCH(\mathcal{G}_{k_i}) \)**
      Based on Lemma 7 and Function NotAllOnConvexHull, robot \( r_j \) moves to state **Compute.OnStraightLine**.

      - **\( r_j \) is in the middle of two other robots that \( \in onCH(\mathcal{G}_{k_i}) \)**
        Based on Lemma 11 and Function OnStraightLine, robot \( r_i \) moves to state **Compute.SeeTwoRobots**. Based on Procedure SeeTwoRobots and per Lemma 13, it returns a point \( p \) with direction away from the convex hull. This case is not possible to happen since it is considered as bad configuration of Type 2.

      - **\( r_j \) is not in the middle of two other robots that \( \in onCH(\mathcal{G}_{k_i}) \)**
        Based on Lemma 11 and Function OnStraightLine, robot \( r_j \) moves to state **Compute.SeeOneRobot**. Procedure SeeOneRobot based on Lemma 12 returns \( c_j \), hence \( r_j \) does not cause \( CH(\mathcal{G}_{k_{m-1}}) \) to change.

    - **\( r_j \) is not on straight line with any two other robots that \( \in onCH(\mathcal{G}_{k_i}) \)**
      Based on Lemma 7 and Function NotAllOnConvexHull, robot \( r_j \) moves to state **Compute.NotOnStraightLine**.

      - **\( r_j \) sees that there exist enough space for at least one robot on \( CH(\mathcal{G}_{k_i}) \)**
Based on Lemma 8 and Function NotOnStraightLine, robot \( r_j \) moves to state Compute.SpaceForMore. Procedure SpaceForMore based on Lemma 9 returns \( c_j \) or \( p \) a point outside of the convex hull if \( r_j \) touches another not adjacent robot on \( onCH(G_{R_{m-1}}) \). If it returns \( c_j \), \( r_j \) does not cause \( CH(G_{R_{m-1}}) \) to change. The case that \( r_j \) touches another not adjacent robot on \( onCH(G_{R_{m-1}}) \) is impossible because this means that the two robots that are touching block at least one robot from seeing other robots, hence it is impossible to have fully visible and this situation.

- \( r_j \) sees that not enough space exists for at least one robot on \( onCH(G_{R_k}) \).

Based on Lemma 8 and Function NotOnStraightLine, robot \( r_j \) moves to state Compute.NoSpaceForMore. Procedure NoSpaceForMore based on Lemma 10 returns a point \( p \) with direction away from the convex hull.

This case is not possible to happen since it is considered as bad configuration of Type 1.

- \( r_j \notin onCH(G_{R_k}) \).

Based on Lemma 3 and Function Start, \( r_j \) moves to state Compute.NotOnConvexHull.

* \( r_j \) is touching another robot.

Based on Lemma 14 and Function NotOnConvexHull, \( r_j \) moves to state Compute.IsTouching. Procedure IsTouching based on Lemma 15 returns a point \( p \in onCH(G_{R_k}) \) or \( c_j \). This case is not possible to happen, because if \( r_j \) did not arrived to \( p \) before \( e_m \), it is not possible for \( |onCH(G_{R_{m-1}})| = n \), since no robot that belongs to \( onCH(G_{R_{m-1}}) \) moves and neither does \( r_j \).

* \( r_j \) is not touching any other robot.

Based on Lemma 14 and Function NotOnConvexHull, \( r_j \) moves to state Compute.NotTouching.

- \( r_j \) can move to \( onCH(G_{R_k}) \) without causing it to change

Based on Lemma 17 and Function NotTouching, \( r_j \) moves to state Compute.NotChange. Based on Lemma 19 Procedure NotChange returns a point \( p \in onCH(G_{R_k}) \). This case is not possible to happen, because if \( r_j \) did not arrived to \( p \) before \( e_m \), it is not possible for \( |onCH(G_{R_{m-1}})| = n \), since no robot that belongs to \( onCH(G_{R_{m-1}}) \) moves and neither does \( r_j \).

- \( r_j \) cannot move to \( onCH(G_{R_k}) \) without causing it to change

Based on Lemma 17 and Function NotTouching, \( r_j \) moves to state Compute.ToChange. Based on Lemma 18 Procedure ToChange returns a point \( p \in onCH(G_{R_k}) \) or \( c_j \). This case is not possible to happen, because if \( r_j \) did not arrived to \( p \) before \( e_m \), it is not possible for \( |onCH(G_{R_{m-1}})| = n \), since no robot that belongs to \( onCH(G_{R_{m-1}}) \) moves and neither does \( r_j \).

(B) \( r_i \) is in a trajectory \((start,target)\), that was decided on a robot configuration, say \( R_k \). It follows that \( k < x \).

This is a similar case with \( r_j \) in 1-A-b.

2. \( e_{m-1} \) on \( r_j \) (indirect)

This is the same case with \( r_j \) in 1-A-a and 1-A-b.
Liveness Properties

Lemma 22 Given any good execution of the algorithm, there exists a configuration $R_m$ such that $|onCH(G_{R_m})| = n$ and $G_{R_m}$ is a fully visible configuration.

Proof: If $R_0$ has the stated properties, there is nothing to prove. So consider the case that $R_0$ is either c1: $|onCH(G_{R_0})| < n$ or c2: $|onCH(G_{R_0})| = n$ and $G_{R_0}$ is not a fully visible configuration.

Based on Lemma 20 if c1 or c2 is true, then $onCH(G_{R_0})$ can only expand, hence $onCH(G_{R_0})$ will not shrink unless c1 and c2 are not true.

We first list the various cases to be considered and then we show how they are interleaved.

1. c1 is true.

   A There exists space for at least one robot to be on the convex hull.
      i Robots that $\in onCH$
         In this case, based on Lemmas 9, 12 and 13 the robots that $\in onCH$ do not move or move outside of the convex hull.
      ii Robots that $\notin onCH$
         a Robots that are tangent with other robots.
            In this case, based on Lemma 15 Robots that are tangent with other robot either stay in the same position, or move to $onCH$.
         b No point on $onCH$ exists, such that Function $\text{FindPoints}$ will return it as valid point.
            In this case, based on Lemma 18 robots that called Function $\text{FindPoints}$ and no point was returned, will move to $onCH$.
         c At least a point on $onCH$ exists, such that Function $\text{FindPoints}$ will return it as valid point.
            In this case, based on Lemma 19 robots that called Function $\text{FindPoints}$ and at least a point was returned, will move to $onCH$.

   B No space exists for at least one robot on $onCH$.
      i Robots that $\notin onCH$.
         In this case, based on Lemmas 15, 18 and 19 robots that $\notin onCH$ do not move.
      ii Robots that $\in onCH$.
         In this case, robots that $\in onCH$, Based on Lemmas 10, 12 and 13 can only move with direction outside of the convex hull or stay at the same position.

2. c2 is true.

   In this case $|onCH(G_{R_0})| = n$ and $G_{R_0}$ is not a fully visible configuration. This implies that at least three robots are on the same line, hence we get the following cases:

   A Robots that are not on a straight line with any two other robots.
      In this case, based on Lemma 9 robots stay in the same position.
   B Robots that are on the same straight line with at least two other robots but are not in the middle of any two other robots that are on the same line.
      In this case, based on Lemma 12 robots stay in the same position.
   C Robots that are on the same straight line with at least two other robots and are in the middle of any two other robots that are on the same line.
      In this case, based on Lemma 13 robots move outside of the convex hull.
We now discuss how the cases above are combined to yield the claimed result.

(a) If no space for at least one robot on the convex hull exist, this is case 1-B. In case 1-B, necessary some robots are on the convex hull and this is case 1-B-ii for some robots. Therefore, robots of case 1-B-ii will continue to expand until a space for at least one robot exists. Hence if a space does not exist, eventually a space for more robots on the convex hull will be created.

(b) If some robots that are touching are in case 1-A-ii-a, based on Lemma 16, at least one robot will move. Hence, eventually the robots that were tangent will no longer be tangent in the same place.

(c) If three or more robots are on the same line, it means that at least one robot is in the middle of two other robots. The robot that is in the middle, based on Lemma 13 will move to the outside of the convex hull. Each time the robots that are not in the middle, Based on Lemma 12 will stay in the same position. Therefore, eventually no three robots will be on the same line and each time there exists a line, the convex hull expands.

(d) If c1 is true it means that at least one robot is not on the convex hull. If a space for at least one robot on the convex hull exists, then it could be one of the cases 1-A-ii. Robots in cases 1-A-ii-a (at least 1), 1-A-ii-b and 1-A-ii-c try to move on the convex hull. If at least one space on the convex hull exists, one of the robots that are inside the convex hull will move to onCH. Because of (c) eventually no three robots will be on the same line, hence the robots on the convex hull will be run Procedure NoSpaceForMore (see the possible cases if c1 is true and no 3 robots are on the same line). If no space exists on the convex hull robots that are on the convex hull will move to expand to the convex hull and create more space as described earlier in (a). Hence if c1 is true it follows that the convex hull expands.

(e) Based on Lemma 2 for any two adjacent robots with centers c_l and c_r, c_l and c_r ∈ onCH, there exists a safe distance between c_l and c_r for which a third robot r_i can be on onCH between c_l and c_r without causing it to change.

(f) Based on (d) and (e) it follows that if c1 is true convex hull will continue expanding and the number of robots that are on the convex hull will increase, until c1 is not true or the safe distance was reached. Some robots that get on the convex hull cause some other robots to no longer be on the convex hull. This means that the convex hull will continue to expand if c1 is true and after a safe distance between neighboring robots on the convex hull is reached, the next robot that is inside the convex hull can and will move on the convex hull without causing it to change. This will continue happening until all robots are on the convex hull. Hence c2 will be true.

(g) If c2 is true, it means that at least three robots are on the same line. Based on (c) the convex hull expands and eventually no three robots will be on the same line. Some robots that move to the outside of the convex hull may cause others to no longer be on the convex hull. Then c1 will be true and based on (f) c2 will be true again. This will continue happening until safe distance is reached (The convex hull continues expanding if c1 or c2 is true). In the same way as in (f) all robots will be on the convex hull without any changes caused and based on (c) no three robots will be on the same line. Hence, all robots will be on the convex hull and all robots will have full visibility. This completes the proof.

The following lemma states that starting from any initial configuration, when the robots form a configuration such that all robots are on the convex hull and they can see each other, then the robots will eventually form a connected configuration.

**Lemma 23** Given any good execution of the algorithm, if \( \mathcal{R}_l \) is such that \(|onCH(\mathcal{G}_{\mathcal{R}_l})| = n \) and \( \mathcal{G}_{\mathcal{R}_l} \) is a fully visible configuration and not a connected configuration, then there exists \( \mathcal{R}_k, l \leq k \) so that \( \mathcal{R}_k \) is a connected configuration.

**Proof:** Based on Lemma 21, if a configuration \( \mathcal{R}_m \) is such that \(|onCH(\mathcal{G}_{\mathcal{R}_m})| = n \) and \( \mathcal{G}_{\mathcal{R}_m} \) is a fully visible configuration, then \(|onCH(\mathcal{G}_{\mathcal{R}_{m+1}})| = n, \mathcal{G}_{\mathcal{R}_{m+1}} \) is a fully visible configuration and \( CH(\mathcal{G}_{\mathcal{R}_{m+1}}) \subseteq CH(\mathcal{G}_{\mathcal{R}_m}) \).
Based on Procedure \textit{NotConnected} (see first three cases of procedure), no robot will start moving unless: Between any three adjacent robots on the convex hull, say \(r_l, r_m, r_r\) left robot, middle robot and right robot respectively, the distance between line segment \(\overline{r_l r_r}\) and \(r_m\) must be equal or more than \(\frac{1}{n}\). This, along with Lemma 22 guarantee that no robot will move unless the distance of \(\frac{1}{n}\) at least exists and that eventually all robots will be on the convex hull and have full visibility. Because no robot moves unless the distance of \(\frac{1}{n}\) at least exists, all robots will eventually move to the \textbf{Look} state and see that the configuration they see is \textit{fully visible} and \(|onCH(V_i)| = n\). We get the three following cases:

A There exists at least one component (as it was defined in Function 3.4) that is smaller than at least one other component, with respect to the number of the robots that consist each component.

Function \textit{NotConnected} results all robots of the smallest component(s) to join one component that is larger than it. Given the liveness condition that whenever a robot decides to move, it moves at least a distance of \(\delta\), eventually the number of the components become smaller and eventually the convex hull shrinks. Also the robots, of the components that are not the smallest, do not move.

B All components are of the same size, with respect to the number of the robots that consist each component. The distance between two neighboring components is not the same for all the neighboring components.

Function \textit{NotConnected} results that all robots of the component that has the smallest distance to its neighbor component on the right to join the component on its right. Given the liveness condition that whenever a robot decides to move, it moves at least a distance of \(\delta\), eventually the number of the components become smaller and eventually the convex hull shrinks. The robots of the other components do not move.

C All components are of the same size, and the distance between any two neighboring components is the same.

Function \textit{NotConnected} results that all the components start moving with direction to the inside of the convex hull. Given the liveness condition that whenever a robot decides to move, it moves at least a distance of \(\delta\), it follows that eventually all the components will touch, because the convex hull shrinks.

From the cases above, it follows that either all the robots of any component that has the smallest number of robots (first case) or of any component that has the smallest distance (second case) to its right neighbor will move to its right neighbor until the number of components become one, or the components will move to the inside of the convex hull until all the components touch (third case).

In every case, robot \(r_i\) runs the Procedure \textit{NotConnected}. Hence, per Lemma 6, robot \(r_i\) moves in such a way that it does not cause \(|onCH(G_{R_{m+1}})| < n\) or \(G_{R_{m+1}}\) not to be a \textit{fully visible} configuration. This completes the proof. 

From Lemmas 22 and 23 we get the following.

\textbf{Corollary 24} Given any good execution of the algorithm, there exists \(R_m\) so that \(G_{R_m}\) is a connected and fully visible configuration.

\subsection*{5.2.2 Any Execution}

We now consider any executions, including bad ones.
Lemma 25 Given any execution of the algorithm, if there is a bad execution fragment $\alpha_{bad}$, then eventually a safe configuration $R_{safe}$ is reached, and after a safe configuration there are no longer any bad configurations in the execution until termination.

Proof: There are 2 possible cases:
(a) The adversary deploys a strategy that aims in causing bad configurations as long as it can (i.e., indefinitely if possible).
(b) The adversary, at some point of the execution, stops causing bad configurations.

We focus on the first case and we show that any execution under this adversarial strategy will eventually reach a configuration in which the adversary will no longer be able to cause bad configurations. It is easy to see that this case covers also the second case.

Recall that both types of bad configurations involve configurations in which the robots are momentarily in a configuration in which all robots are on the convex hull and it is fully visible, but the adversary manages to break this property. The adversary, as explained, exploits the fact that some robots, due to asynchrony, are not aware that such a configuration has been reached. We now consider the two types of bad configurations.

(i) Bad configuration of type 1. Consider the case in which the first bad configuration, call it $R_x$, that appears in the bad execution fragment $\alpha_{bad}$ is of type 1 (the other type is considered later). As explained, the adversary may deploy a strategy which can result into a configuration $R_z$, $z > x$, so that $G_{R_z}$ is no longer fully visible or and not all robots are on the convex hull. The adversary can do so, if there is at least one robot that according to its local view in configuration $R_x$, not all robot are on the convex hull and there is no more space for an “internal” robot to get on the convex hull (per Function NoSpaceForMore this robot will move to a direction outside of the convex hull). It follows that $CH(G_{R_z}) \supseteq CH(G_{R_x})$. Furthermore, from Lemma 20 we get that for all successive configurations of $R_z$ in which not all robots are on the convex hull or are fully visible, the convex hull can only expand (until a configuration in which these properties hold is reached). The adversary may repeat this strategy (e.g., involving other robots on the convex hull), every time causing the convex hull to expand. However, per Lemma 2 this cannot be repeated indefinitely, as the convex hull will expand that much, that the safe distance will be reached for all pairs of adjacent robots on the convex hull. From this and the liveness condition (the adversary must allow a robot to move by at least $\delta$ distance) it follows that a configuration is eventually reached after which no bad configuration of type 1 can exist (no robot will get into state Compute.NoSpaceForMore). Observe that when such a configuration is reached, it is still possible for a bad configuration of type 2 to be reached. This is covered by the next case we consider (with the difference that this bad configuration is not the first appearing in $\alpha_{bad}$).

(ii) Bad configuration of type 2. Consider the case in which the first bad configuration, call it $R_x$, that appears in the bad execution fragment $\alpha_{bad}$ is of type 2. This is the situation where in a preceding configuration there are at least four robots on a straight line on the convex hull. As explained in Section 5.1 the adversary can yield a configuration in which not all robots are any longer on the convex hull, or there is no full visibility. However, per Function SeeOneRobot and Lemma 12 the robots on the straight line that are not in the middle (i.e., they see only one robot) do not move. In contrast, according to Function SeeTwoRobot and Lemma 13 each robot in the middle of the straight line moves in a direction outside of the convex hull, in such a way that it will no longer be in a straight line with its two adjacent robots (on the convex hull). It follows that if every time the adversary repeats the same strategy, and say initially there are $x$ robots on straight line, then in every iteration the number of robots that are on the same line is $x - 2$. This may continue only until $x$ is less than 3, hence it eventually stops. Observe that during these iterations, since robots in the middle move towards a direction outside of the convex hull and per Lemma 20 the convex hull can only expand. Hence a bad configuration of type 2 can no longer exist. Furthermore, note that if during this expansion, the robots involved have also reached the safe distance (per Lemma 2's definition), then as explained above, a bad configuration of type 1 also cannot exist. Otherwise, we are back in case (i) as discussed above. Note however that once robots reach the safe distance, and a bad configuration of type
2 is reached, a configuration of type 1 can no longer exist again: when a robot has already safe distance between its adjacent robots on the convex hull, then the middle robots by moving towards outside the convex hull can only increase the safe distance (and hence it will not be possible for a robot to get into state \texttt{Compute.NoSpaceForMore}).

From cases (i) and (ii) and Lemma \ref{lemma:safe} it follows that a fully visible configuration in which $\lvert onCH \rvert = n$ is reached. By a similar argument as in the proof of Lemma \ref{lemma:safe} we get that eventually a safe configuration is reached (all robots are on the convex and they are aware that the configuration is fully visible). From Function $\text{NotConnected}$ and Lemma \ref{lemma:fully_visible} it follows that any succeeding configuration maintains the property that all robots can see each other and that are on the convex hull. Hence, the algorithm is such that once a safe configuration is reached, it is no longer possible for a bad configuration to exist. This completes the proof.

We are now ready to prove that our algorithm solves the gathering problem.

\textbf{Theorem 26 (Gathering)} In any execution of algorithm, there exists a configuration $R_m$, so that $G_{R_m}$ is a connected, fully visible configuration and $\forall s_i \in S_{R_m}$, $s_i = \text{Terminate}$.

\textbf{Proof:} Consider the following two cases.

- If no bad configurations exist, based on Corollary \ref{corollary:connected}, given any good execution of the algorithm, there exists $R_m$ so that $G_{R_m}$ is a connected and fully visible configuration.

- If bad configurations exist, based on Lemma \ref{lemma:bad configurations}, given any execution of the algorithm, if there is a bad execution fragment $\alpha_{bad}$, then eventually a safe configuration $R_{safe}$ is reached, and after a safe configuration there are no longer any bad configurations in the execution until termination. Therefore, from this point onward, we get from Corollary \ref{corollary:connected} that there exists $R_m$ so that $G_{R_m}$ is a connected and fully visible configuration.

When a connected and fully visible configuration is reached, it is easy to see that robots no longer move and eventually all robots get into state \texttt{Compute.Connected} and hence into state \texttt{Terminate}.

\section{Conclusions}

In this paper we have considered the problem of gathering non-transparent, fat robots in an asynchronous setting. We have formulated the problem and the model with a state-machine representation and developed a Distributed Algorithm that solves the problem for any number of robots. The correctness of our algorithm relies on the assumption of chilarity \cite{10} (robots agree on the orientation of the axes of their local coordination system). This is the only assumption we needed to add to the model considered in \cite{8}. We believe this is a very small price to pay in order to solve the gathering problem for any number of fat robots. Nevertheless, it would be very interesting to investigate whether one can remove this assumption and still be able to solve the gathering problem for any number of fat robots. Certainly one will need to take a different approach than the one we use in this paper, as our approach depends greatly on this assumption.

\section*{References}

[1] N. Agmon and D. Peleg. Fault-tolerant gathering algorithms for autonomous mobile robots. In \textit{Proc. of the 15th ACM-SIAM Symposium on Discrete Algorithms (SODA 2004)}, pages 1070–1078.
[2] H. Ando, Y. Oasa, I. Suzuki, and M. Yamashita. Distributed memoryless point convergence algorithm for mobile robots with limited visibility. *IEEE Transactions on Robotics and Automation*, 15(5):818–828, 1999.

[3] H. Attiya and J. Welch. *Distributed Computing: Fundamentals, Simulations and Advanced Topics*. Second edition, Wiley & Sons, 2004.

[4] A. Bandettini, F. Luporini, and G. Viglietta. A survey on open problems for mobile robots. In *arXiv:1111.2259v1*, 2011.

[5] K. Bolla, T. Kovcs, and G. Fazekas. Gathering of fat robots with limited visibility and without global navigation. In *Proc. of ICAISC/SIDE-EC 2012*, pages 30–38.

[6] S.G. Chaudhuri and K. Mukhopadhyaya. Gathering asynchronous transparent fat robots. In *Proc. of the 6th International Conference on Distributed Computing and Internet Technology (ICDCIT 2010)*, pages 170–175.

[7] A. Cord-Landwehr, B. Degener, M. Fischer, M. Hüllmann, B. Kempkes, A. Klaas, P. Kling, S. Kurras, M. Märtens, F.M.A Der Heide, C. Raupach, K. Swierkot, D. Warner, C. Weddemann, and D. Wonisch. Collisionless gathering of robots with an extent. In *Proc. of the 37th International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM 2011)*, pages 178–189.

[8] J. Czyzowicz, L. Gasieniec, and A. Pelc. Gathering few fat mobile robots in the plane. *Theoretical Computer Science*, 410(6–7):481–499, 2009.

[9] A. Dutta, S. G. Chaudhuri, S. Datta, and K. Mukhopadhyaya. Circle formation by asynchronous fat robots with limited visibility. In *Proc. of the 8th International Conference on Distributed Computing and Internet Technology (ICDCIT 2012)*, pages 83–93.

[10] P. Flocchini, G. Prencipe, and N. Santoro. *Distributed Computing by Oblivious Mobile Robots*. Synthesis Lectures on Distributed Computing Theory, Morgan & Claypool Publishers, 2012.

[11] P. Flocchini, G. Prencipe, N. Santoro, and P. Widmayer. Gathering of asynchronous robots with limited visibility. *Theoretical Computer Science*, 337(1–3):147–168, 2005.

[12] R.L. Graham. An efficient algorithm for determining the convex hull of a finite planar set. *Information Processing Letters*, 1(4):132–133, 1972.

[13] S. Souissi, T. Izumi, and K. Wada. Distributed algorithms for cooperative mobile robots: A survey. In *Proc. of the 2nd Second International Conference on Networking and Computing (ICNC 2011)*, pages 364–371.