Assembling of Robots in Presence of Line Obstacles with Direction-only Axes Agreement

A Sadhu¹ And S Mukhopadhyaya²
¹Dr. B C Roy Engineering College Durgapur, Jemua Road, Fuljhore, Durgapur, West Bengal, India.
²Birla Institute of Technology, Mesra, Lalpur extension center, Lalpur, Ranchi, Jharkhand, India.
E-mail: arun.sadhu@bcrec.ac.in

Abstract. This paper addresses the problem of assembling semi-synchronous oblivious robots in presence of horizontal line obstacles under direction only axes agreement model. In this article, a distributed algorithm is proposed for a swarm of autonomous mobile robots that are required to assemble over the boundaries of a rectangular bounded region within a finite amount of time. The robots are initially deployed randomly within the region and assumed to support the non-rigid motion. All the line-shaped obstacles are randomly scattered within the region and are assumed to be positioned horizontally, parallel to the top and bottom boundaries of the rectangular region. Though the robots are assumed to have unlimited visibility, these opaque obstacles restrict their vision. The existing solution to this problem assumes full axes agreement among the robots, whereas, the proposed solution assumes direction only axis agreement, a much weaker model. Our proposed algorithm is fault-tolerant and supports the collision-free movement of the robots. It successfully assembles robots on the left and right boundaries of the region.

1. Introduction
The swarm robotic research considers a group of very simple, autonomous robots who collectively perform various tasks by their continuous cooperation and coordination. Each robot is capable of observing its surroundings, carrying out simple computations, and moving freely over the plane. This behavior of the robots is designed by observing the activities of insects, birds, etc. for their survival. Some of these behaviors like foraging, vigilance, flight, etc. help researchers to design optimization algorithms [1], [2], [3], as well as, to design a low-cost multi-robot system [4]. Multi-robot systems are useful in various applications like exploration, rescue operation, target searching, surface cleaning, etc. where human intervention is difficult. Researchers usually design distributed algorithms to solve basic problems like pattern formation, flocking, partitioning, searching, gathering, convergence, coverage, etc. by a swarm of robots [5],[6],[7],[8],[9],[10],[11].

Assembling a swarm of robots on a line or any other shape can be considered as a preliminary step to solving various complex problems, like, area partitioning [12], spreading [13], etc. In the spreading problem, robots are required to position themselves uniformly over a line from their random initial configuration over a plane. However, even assembling all the robots on that line is itself a challenging task. The self-assembling of robots on a line may be considered as a pre-processing step in this scenario. The objective of the assembling problem is to gather all the robots over a particular shape within a finite amount of time [14]. Robots are initially deployed over the given work-space at random. The work-space is usually considered to be free of any obstacles, which is not quite realistic.

The presence of obstacles in the work-space not only creates barriers in the paths followed by the robots, but it also obstructs the mobility and visibility of the robots as well [15]. Thus, it adds a different
dimension to the classical problems. Robots usually plan their paths by avoiding obstacles coming in their way. Some of the recent works [4], [16] use probabilistic algorithms to plan the path by avoiding obstacles. Robots, when encountering obstacles of different shapes on their way, cannot restore their formation and partition themselves into smaller groups to avoid the obstacles. Once the obstacle is avoided, the robots again unite and regain the formation [17].

In this paper, we have addressed the problem of assembling in presence of obstacles. Initially, the robots are assumed to be randomly deployed over a finite rectangular region, which contains some opaque, line obstacles. These obstacles are assumed to be placed horizontally, that is, in parallel to the top and bottom boundaries of the rectangular region. The robots are required to assemble themselves on the left and right boundaries of the region in a finite amount of time. Our proposed solution is crash fault-tolerant. We ensure paths followed by the robots during assembling are collision-free.

The paper is organized into five sections. After discussing the current state of the art, our contribution in this paper is mentioned in the second section. The problem definition, models, and basic assumptions are discussed in the third section. The proposed assembling algorithm for direction only axis agreement and its correctness is presented in the fourth section. In this section, an upper bound on the number of active cycles has also been estimated. Finally, section 5 concludes the paper.

2. Related works

Axis agreement among the robots has a significant impact on the solvability of a problem. The full compass axis agreement indicates that the robots at the work-space agree on both the direction and orientation of their local axes while in the half compass agreement, robots agree on the direction of both the axes but orientation of only one of the axes. Efriama et. al., in their paper [18] discussed various types of axis agreement and their effect on the robots.

Due to the naive capacity of the robots, they might become non-operational or faulty during the execution of an algorithm. Therefore, fault tolerance is also an important aspect. Defago et. al. in their paper [19] presented a detailed discussion on various fault models available in the literature and the role of faulty robots in solving various problems. Restricted visibility of robots is also another down to earth assumption made by the researchers [9],[20],[21],[22] while proposing solutions to different classical problems. The restrictions considered are either due to the presence of opaque obstacles falling on the line of sight or due to the limited capacity of the sensors. In case, sensors are not capable enough to sense the whole work-space, the visibility graph of the robots is usually assumed to be connected for the sake of solvability of a problem. Under limited visibility, robots are assumed to be able to see up to a certain distance, say, $\delta (> 0)$ unit. Path planning problem for mobile robots is also an important basic problem when obstacles are assumed to be present in the workplace [23],[24],[25]. It is desirable that the mobile robots should follow paths that avoid obstacles and are collision-free, as well. In assembling problem, robots usually assemble themselves by forming a pattern autonomously. They start from their initial scattered positions and collectively move towards forming a pre-specified pattern within a finite amount of time [14],[26].

The presence of obstacles within the workspace obstructs robots’ mobility and visibility, which makes the task more difficult. The assembling problem in presence of obstacles is first addressed by Das et al. [27]. In this work, the authors have considered asynchronous robots to assemble on a boundary of a finite rectangular region, containing horizontal, line obstacles inside it. However, in this algorithm, the robots are assumed to have full compass axes agreement and unlimited visibility. Das et al. extended their work [28] by proposing a solution to the assembling problem in presence of the same horizontal line obstacles with semi-synchronous robots having limited visibility. Here also the authors assume full compass axis agreement among the robots in the swarm.

2.1. Our Contribution

The assembling problem in presence of opaque, horizontal, line obstacles is addressed in this paper. In the proposed solution, the robots are assumed to have direction only axis agreement, a much weaker model, closer to reality than the one assumed in the existing literature [27], [28]. The robots here are
assumed to be very simple, anonymous, oblivious, semi-synchronous, and support the non-rigid motion. They are required to assemble and occupy distinct positions on the left and right boundaries of the given work-space within a finite amount of time. Moreover, the proposed solution is crash fault-tolerant. In the entire process, the movements of the robots are collision-free.

3. Problem Definition and Assumptions

In this section, let us first formally define the problem addressed in the paper.

Assembling Problem

Robots are assumed to be deployed randomly within a given finite rectangular region. Some horizontal line obstacles are assumed to be scattered randomly inside the region. Positions of the obstacles inside the region are fixed, but not known to the robots. However, because of the unlimited vision, the robots can see these obstacles if there is no obstruction in their line of sight. The robots are required to assemble on distinct positions along the left and right boundaries of the region.

The assumptions made regarding the environment, models, and characteristics of robots are listed as follows:

Internal Environment

- Robots are randomly deployed over a rectangular bounded region so that no two robots occupy the same position. The boundaries of the target region are assumed to be parallel to the local axes of the robots.
- Horizontal (parallel to the top and bottom boundaries of the region), line obstacles of negligible width are scattered over the region.
- There is a minimum separation $\lambda > 0$ between any two objects, robots, and obstacles.
- Positions of the obstacles are fixed throughout the process.
- The obstacles are opaque and hence obstruct the vision of a robot if they come on its line of sight.
- None of the obstacles touch the left and right boundaries of the target region.
- The internal environment is not known to the robots a priori.

Models

For the proposed algorithms the following models are assumed.

Computational model:
Each robot is considered as a small computational device. The CORDA model is used here to set the mode of computation of the robots. In this model, robots are assumed to execute a sequence of computational cycles until they complete their jobs. Each of these cycles consists of three phases, (i) look: robots look around to take a snapshot of their surroundings in search of the positions of the other robots in the region, (ii) compute: robots compute their next destination using snapshot collected in look phase. And (iii) move: robots move towards their destination as computed in the compute phase.

Synchrony:
In the proposed algorithms robots are assumed to be synchronous or semi-synchronous.
- Synchronous (FSYNC): Robots in the swarm follow a global clock to execute the phases of its computational cycles. All the robots in the swarm execute the same phases at the same time.
- Semi-synchronous (SSYNC): All robots are not necessarily alive always. However, only alive robots act synchronously.

Axes agreement:
All the robots in the swarm maintain a local coordinate system, of which, the robot itself is occupying the position $(0, 0)$. In this work, it is assumed that the robots are having direction only axes agreement
among themselves. That is, the directions of both the local axes of the robots are identical for all but orientation may differ.

Robot Characteristics

The autonomous and anonymous robots in the swarm are assumed to have the following characteristics: Robots are oblivious and silent. Neither do they retain any past memory nor do they have direct communication among themselves through message passing, etc. Robots are assumed to have unlimited visibility. However, due to the presence of opaque objects within the target region, their line of sight may get obstructed. As robots are considered to be point robots, they do not obstruct the line of sight of each other.

Robots are assumed to have non-rigid motion; a robot can stop anywhere in between, before reaching its computed destination. However, for the sake of finiteness, it is guaranteed that in each cycle a robot must travel at least a small finite distance, δ (> 0), towards its destination, where, δ is a finite quantity, δ < λ.

4. Proposed Algorithm for Assembling Problem under Direction-only model

In this section, an assembling algorithm is proposed which is applicable for the direction-only model. Since the full-compass and half-compass axes agreement is covered by direction-only axis agreement, the proposed algorithm is applicable for both full-compass and half-compass models as well.

The existing algorithm in the literature by Das et. al. [27] when applied under a half-compass or direction-only model, may not give any solution at all. Moreover, robots would end up landing on either boundary of the region, left or right, as they do not agree on the orientation of their x-axis. However, even if we accept the robots to gather on either side of the region, the algorithm doesn't even successfully terminate in some scenarios. The adverse situations that arise when the existing algorithm [27] is applied on the half-compass axes agreement model can easily be handled in a conflict-free way to assemble the robots by the algorithm proposed in this section.

For ease of understanding, let us mention the target rectangular region as ABCD, as shown in Figure 1. The proposed algorithm assembles the robots on either side of the region, that is, on left boundary AD or right boundary BC of the region ABCD. A brief outline of the proposed algorithm Assemble_direction_only is as follows:

Throughout the algorithm, a robot determines its action according to the following three cases:

Case A: If a robot R identifies itself on the left or right boundary of the region, it terminates. Otherwise,

Case B: If R finds that the horizontal line (along its local x-axis) passing through itself towards the left or the right boundary, whichever nearer, is free of any object, robot, or obstacle, then R moves directly
to that boundary following that horizontal path. The procedure terminates when \( R \) reaches the boundary, left or right. If both the boundaries are at the same distance, it selects that boundary towards which the horizontal path is free. If both sides are free, \( R \) moves to the left (according to its orientation) boundary. Otherwise,

**Case C:** If any other object, obstacle or robot, other than \( R \), is found to be present on the horizontal line passing through \( R \) towards the nearer boundary, then \( R \) first identifies an object (an obstacle and/or a robot and/or a boundary), say \( P \), above (according to its local coordinate system, in the positive direction of the y-axis) its horizontal level, vertically closest to itself. Depending on the position of \( P, R \) identifies the decision rectangle \( XYZW \), as shown in Figure 1. All the sides of the decision rectangle are parallel to the axes of the local coordinate system of the respective robot. The side \( XY \) passes through the robot \( R \) and \( WZ \) passes through the object \( P \). The other two vertical sides \( WX \) and \( ZY \) are part of the left and right boundaries of the target region \( ABCD \) respectively. \( WZ \) is considered as the upper boundary of the target region in absence of any object, such as obstacles or robots placed between the top (concerning the respective robot) boundary and robot \( R \) itself. Let \( T \) be the point of intersection of the vertical line passing through \( R \) with one of the two diagonals \( XZ \) and \( WY \), whichever is vertically nearer to \( R \). Depending on the position of \( R \) with respect to the point \( O \) (point of intersection of the two diagonals), the point \( T \) can either be on \( WY \) or \( XZ \) as shown in Figure 1. If \( T \) coincides with \( O \), \( R \) moves horizontally towards its negative x-direction through a very small distance \( \epsilon (> 0) \), where, \( \delta \leq \epsilon < \lambda \); as \( \delta, \lambda \) are defined in Sec.3. Otherwise, \( R \) moves directly to \( T \).

A formal description of the proposed algorithm is given below. In the description of the algorithm by the terms left and right it is meant the negative and the positive direction of the local (to the robot) x-axis respectively.

**Procedure** Assemble_direction_only

**Phase Observe:**

Robot \( R \) takes a snapshot of its surroundings.

**Phase Compute:**

Robot \( R \) executes the following

if (\( R \) identifies itself on either right or left boundaries)

\( R \) sets its next destination at \((0,0)\) and terminates.

else

if (\( R \) finds no other obstacles or robots along its local x-axis towards the boundary, left or right, whichever nearer)

\( R \) sets its next destination at \((d, 0)\), where \(|d|\) is the horizontal distance of \( R \) from the left or right boundary, whichever nearer.

if (Both the boundaries are at the same distance)

\( R \) sets its destination on that boundary towards which the horizontal path is free.

endif

if (Both the paths are free)

\( R \) selects the left one and fixes the destination at \((-|d|, 0)\).

endif
else

\( R \) identifies the decision rectangle and identifies its destination at point \( T \) as discussed above.

if (\( T \) coincides with \( O \))

\( R \) sets the destination at \((-\epsilon, \ 0)\)

endif
endif
endif

Phase Move:

Robot \( R \) moves to its computed destination.

4.1. Correctness of the Algorithm

In this section, we establish that the proposed assemble algorithm assembles all the synchronous/semi-synchronous robots in the distinct positions on the left and right boundaries of the target region under direction only model in a finite time, following non-rigid motion. Moreover, the paths followed by the robots are all collision-free. For the ease of discussion, let us consider a pseudo partition of the target region made by horizontal lines (virtual) passing through each obstacle. The sub-regions generated by these virtual lines are termed blocks. Figure 2 shows such a partition. These blocks are fixed as the positions of the obstacles are fixed. In the following discussion, the directional terms, like, upper, lower, right, and left, when used with respect to a global notion, the term global is used. Otherwise, the directions are meant to be local, with respect to the concerned robot.

![Figure 2. Working space is partitioned into blocks](image)

**Observation 1.** Throughout the algorithm, a robot either moves horizontally towards the boundary, left or right, whichever nearer to its initial position or it moves in a vertically upward direction with respect to its local coordinate system.

**Observation 2.** In a particular computational cycle, a robot, if at all moves in a vertical direction, does not move more than half the distance from its vertically nearest object in the upward direction.

A robot takes a vertical move only if it sees an obstacle or another robot present on its way to the nearer boundary (left or right) through a horizontal path passing through its current position. In the vertical movement, \( R \) sets its destination at \( T \) and follows the path \( RT \). The point \( T \) is always below the
horizontal level of $O$, which is at the middle of the vertical distance between $R$ and $P$. All these points have the usual meaning as discussed above and can be referred to in Figure 2 and Figure 5.

**Observation 3.** A robot, during its vertically upward movement (say, in a computational cycle $C_i$) does not reach the level of the next robot above itself (as observed in $C_i$), even if they have the opposite orientation.

If two robots have opposite orientations, through vertical movement they may come closer to each other. However, they cannot reach the level of the other because of observations 1 and 2. Moreover, to ensure this, in the proposed algorithm, if $T$ coincides with $O$, $R$ changes its position horizontally shifting towards the left and then taking a vertical movement.

**Figure 3.** Scenario explaining observation 3

In Figure 3, both the robots $R_1$ and $R_2$ have the same decision rectangle $WXYZ$, in the opposite orientation. It explains such a situation where robots $R_1$ and $R_2$ (having opposite orientation) may come close to each other when they reach their destination $T_1$ and $T_2$ respectively. Because of Observation 3, we can state the following lemma:

**Lemma 1.** Once a position of a robot satisfies case $B$, its successive positions satisfy either case $B$ or case $A$.

The position of a robot satisfies the conditions of case $C$ only when the horizontal path towards the left/right boundary, whichever nearer, is not free of any obstacle or robot. In this case, $R$ considers the decision rectangle and moves to its identified destination $T$. Thus, even if more than one robot starts their movement from the same level in some computational cycle, say, $C_i$, their projected positions on the lower diagonals is all at distinct heights if they are on the same side of $O$. However, two robots on different sides of $O$ may have their destinations, $T$, on the same height but they aim to move to different boundaries in the next computational cycle. Thus, in $C_{i+1}$, their positions would satisfy case $B$, if they successfully reach $T$. In the case of rigid motion, $R$ is guaranteed to reach $T$ in one cycle. Only in the case of non-rigid motion, in $C_{i+1}$, the robots may still be in a situation satisfying the same case $C$.

**Lemma 2.** The upper boundary of a decision rectangle, considered by $R$, can never exceed the upper boundary of the block $B$, within which the robot $R$ is initially placed.
Proof: In any computational cycle $C_i$, let a robot $R$ be placed within a block $B$, with the upper boundary of the block, say $U_B$. Let us assume that the position of $R$ satisfies case $C$. The upper boundary $U_{DR}$ of the decision rectangle considered by $R$ in $C_i$ is defined either by (i) an obstacle $P$ on $U_B$, whose position is fixed or (ii) by another robot, say, $R'$, below $U_B$ but above $R$. In the first case, $U_{DR}$ coincides with $U_B$ and in the latter case; $U_{DR}$ is below $U_B$ according to $R$'s local coordinate system. Figure 4 and Figure 5 explain these two situations respectively. Now, in both cases, if in the cycle $C_i$, $R$ reaches its computed destination $T$, then the robot $R$ does not need to move any further upward direction throughout the algorithm. In the further computational cycles, it will move horizontally directly to one of the boundaries. However, if $R$ does not reach $T$ in $C_i$ due to non-rigid motion, in the next computational cycle $C_{i+1}$ it may once again need to consider a new decision rectangle. This situation is described in detail as follows:

Let us consider the case (i) above. As positions of the obstacles are fixed, in $C_{i+1}$ if no robots appear between $R$ and $P$, the upper boundary $U_{DR}$ of the decision rectangle remains the same as the upper boundary $U_B$ of the block $B$. If any robot appears in between $R$ and $U_B$, $U_{DR}$ may change in $C_{i+1}$. However, in any case, $U_{DR}$ cannot move beyond $U_B$.

In case (ii), the robot $R'$, if moved vertically upward in $C_i$, can define the new position of the upper boundary of the decision rectangle of $R$ in $C_{i+1}$. However, as $R'$ cannot move beyond $U_B$ (by logic given in case (i), when applied on $R'$), the upper boundary of the decision rectangle of $R$ in $C_{i+1}$ cannot cross $U_B$.

Other than the above two cases, another robot $R''$ which may be in the same level as $R$ but to the right of it in $C_i$ may also define the upper boundary of the decision rectangle of $R$ in $C_{i+1}$. Since the destination of $R''$ in $C_i$ is below $U_{DR}$, $R''$ cannot move beyond $U_{DR}$ in $C_i$. Hence, the upper boundary of the new decision rectangle of $R$ in $C_{i+1}$ cannot be beyond $U_{DR}$ and hence beyond $U_B$.

Thus, if the upper boundary of the decision rectangle of a robot is within the block $B$ in $C_i$, it cannot be shifted beyond the block $B$ in $C_{i+1}$ also. Hence, by induction the lemma is true.

Theorem 1. Assemble_direction_only algorithm terminates in finite time.

Proof: When the robots are synchronous, they are always alive. On the other hand, in the case of semi-synchronous swarms, robots may not be always active. However, a robot cannot be inactive for an indefinite period. Hence, the finiteness of the Assemble_direction_only algorithm can be established if it can be proved that the robots reach the left/right boundary in a finite number of active computational cycles.

In the following discussion, only active computational cycles are counted. A robot $R$ initially placed on any of the two boundaries, takes only one computational cycle to terminate the algorithm. In this case, the initial position of $R$ satisfies case $A$.

Because of Lemma 1, once a robot reaches a position satisfying case $B$, it takes at most $\lceil d/\delta \rceil$ number of active computational cycles to reach the left or right boundary, whichever nearer, where $d$ is its distance from that boundary, and $\delta$ has its usual meaning as described in Sec.1. Hence, a robot whose
initial position satisfies case B, after a finite number of cycles reaches a position satisfying case A. That is, it reaches the left or right boundary in a finite number of computational cycles.

If the position of a robot satisfies the condition of case C, the robot has to take a vertical path to reach its destination $T$, and hence if $R$ can reach $T$, its positions in the next cycles satisfy case B or case A, according to Lemma 1. However, if $R$ stops before reaching $T$, in the successive cycles it may need to consider new decision rectangles.

Now, for a robot $R$, whose initial position satisfies case C, the upper boundaries of the successive decision rectangles cannot go beyond the upper boundary of the block, where $R$ is initially positioned, as discussed in Lemma 2. Hence, after a finite number of steps (cycles), the destination of $R$ (the point $T$ of the current decision rectangle) comes within $\delta$ distance from the starting point. As a result, $R$ successfully reaches its destination $T$. At this point, the position of $R$ satisfies case B. Once it reaches such a position, the rest of the procedure, that is, its horizontal movement is completed in a finite number of steps.

Because of observations 1, 2, and 3, the following theorem can be stated:

**Theorem 2.** The movements of the robots are collision-free.

Proof: A robot satisfying case A does not move at all. One satisfying case B moves only in the horizontal direction through a free path. A robot whose position satisfies case C only takes a vertical movement. According to observations 1, 2, and 3, vertical movements are also collision-free and robots assemble on distinct positions on either boundary.

**Theorem 3.** The algorithm is fault-tolerant.

Proof: Paths followed by the robots to their final destination on the boundaries are all disjoint. Moreover, a robot decides by looking at the positions of other robots. However, a robot’s action does not depend on any action taken by the other robots. Hence, if a robot becomes faulty during the execution of the algorithm, it would not affect the actions taken by other robots. Hence, non-faulty robots can successfully gather on the boundaries.

4.2. **Time Complexity**

The number of computational cycles required by the robots to assemble on the boundaries can be estimated in the case of rigid motion. In the case of non-rigid motion, an upper bound of the same can be estimated. In the case of semi-synchronous robots, these counts are for active cycles only.

In the case of rigid motion, robots whose initial location satisfies case $A$ are already on the boundaries. Robots whose initial locations satisfy case $B$, take only one cycle to reach the boundary (nearer to the respective robots) and one more cycle to terminate the algorithm. Robots whose initial location is described in case $C$ are categorized in two ways: (i) Robots whose horizontal distances from left and right boundaries are the same initially (ii) Robots not satisfying that condition (i). A robot of the first category takes one cycle to change its initial position horizontally towards the left through a very small distance, one cycle to complete its vertical movement, and one more to complete the horizontal movement and reach the boundary. Thus, these robots take four cycles to terminate the algorithm. Robots of the second category thus take three cycles to terminate the algorithm. Thus, all the robots in the swarm assemble on the boundaries within four computational cycles if they follow the rigid motion.

In the case of non-rigid motion, a robot may not reach its computed destination in the move phase. To complete both the vertical and horizontal movement, may require several computational cycles. However, in this case, as a standard model, it is assumed that in each computational cycle a robot moves at least a finite distance of $\min \{\delta, \text{computed distance}\}$ towards its destination. Let $l$ and $b$ be the length and breadth of the target region. The horizontal distance of a robot (in initial distribution) from the nearer of the left or right boundary is at most $\frac{b}{2}$. Hence the number of computational cycles required to complete horizontal movement is at most $\lceil \frac{b}{2\delta} \rceil$. A robot does not need to traverse vertically, a distance more than half of the length of the block where it resides initially. The length of a block can be at most $l$. In the worst case (if all obstacles are placed on the boundaries only), the number of computational cycles
required to complete vertical movement is at most \( \lfloor l/2 \rfloor \). Hence, all robots in the swarm assemble on the boundaries in \( O((l + b)/2) \) active computational cycles if they follow the non-rigid motion.

5. Conclusion
In this paper, solutions to a basic problem, namely, assembling under the presence of obstacles is proposed. Robots are assumed to have unlimited visibility which may get obstructed due to the presence of obstacles. The proposed algorithm is collision-free and supports the non-rigid motion of the robots.

The existing assembling algorithms are designed for full compass axes agreement, whereas, the proposed one is for direction only model, a much weaker and thus wider model. The proposed algorithm terminates after a finite number of active cycles (under both SSYNC and FSYNC model), an upper bound on the required number of active cycles has been estimated. This algorithm is fault-tolerant as well. However, the proposed algorithm assembles the robots on two boundaries instead of one, which may be considered as a limitation of this solution. With direction only axes agreement, assembling robots on only one boundary may be taken up as future work.

References
[1] Ahmed H and Glasgow J 2012 Swarm Intelligence: Concepts, Models and Applications doi: 10.13140/2.1.1320.2568.
[2] Meng X B et al 2016 A new bio-inspired optimisation algorithm: Bird Swarm Algorithm Journal of Experimental & Theoretical Artificial Intelligence 28 4 pp 673–687 doi: 10.1080/0952813X.2015.1042530.
[3] Tan Y and Zheng Z Y 2013 Research Advance in Swarm Robotics Defence Technology 9 1 pp 18–39 doi: 10.1016/j.dt.2013.03.001.
[4] Tan Y ed 2016 Handbook of research on design, control, and modeling of swarm robotics ACIR Book series Hershey PA doi: 10.4018/978-1-4666-9572-6.
[5] Ando H et al 1999 Distributed memoryless point convergence algorithm for mobile robots with limited visibility IEEE Transactions on Robotics and Automation 15 5 pp 818–828 doi: 10.1109/70.795787.
[6] D’Angelo G et al 2014 Gathering and Exclusive Searching on Rings under Minimal Assumptions Distributed Computing and Networking (LNCS 8314) eds Hutchison D et al Springer Berlin Heidelberg pp 149–164 doi: 10.1007/978-3-642-45249-9_10.
[7] Das D and Mukhopadhyaya S 2012 Painting an Area by Swarm of Mobile Robots with Limited Visibility Wireless Networks and Computational Intelligence (CCIS 292) eds Venugopal K R and Patnaik L M Springer Berlin Heidelberg pp 446–455 doi: 10.1007/978-3-642-31686-9_52.
[8] Diederonné Y et al 2010 Leader Election Problem versus Pattern Formation Problem Distributed Computing (LNCS 9888) eds Hutchison D et al Springer Berlin Heidelberg pp 267–281 doi: 10.1007/978-3-642-15763-9_26.
[9] Flocchini P et al 2005 Gathering of asynchronous robots with limited visibility Theoretical Computer Science 337 1–3 pp 147–168 doi: 10.1016/j.tcs.2005.01.001.
[10] Prencipe G 2014 Autonomous Mobile Robots: A Distributed Computing Perspective Algorithms for Sensor Systems (LNCS 8243) eds Flocchini P et al Springer Berlin Heidelberg pp 6–21 doi: 10.1007/978-3-642-45346-5_2.
[11] Turgut A E et al 2008 Self-organized flocking in mobile robot swarms Swarm Intelligence 2 pp 97–120 doi: 10.1007/s11721-008-0016-2.
[12] Acevedo J J et al 2016 A Distributed Algorithm for Area Partitioning in Grid-Shape and Vector-Shape Configurations with Multiple Aerial Robots Journal of Intelligent & Robotic Systems 84 1–4 pp 543–557 doi: 10.1007/s10846-015-0272-5.
[13] Cohen R and Peleg D 2008 Local spreading algorithms for autonomous robot systems Theoretical Computer Science 399 1–2 pp 71–82 doi: 10.1016/j.tcs.2008.02.007.
[14] Yang H A et al 2019 A distributed and parallel self-assembly approach for swarm robotics Robotics and Autonomous Systems 118 pp 80–92 doi: 10.1016/j.robot.2019.04.011.
[15] Baventher U and Das D 2015 Gathering of Swarm of Mobile Robots in Presence of Horizontal Obstacles Intelligent Computing and Applications (AISC 343) eds Mandal D et al Springer New Delhi India pp 49–58 doi: 10.1007/978-81-322-2268-2_6.
[16] Pandey A 2017 Mobile Robot Navigation and Obstacle Avoidance Techniques: A Review International Robotics & Automation Journal 2 3 doi: 10.15406/iratj.2017.02.00023.
[17] Dasgupta P and Cheng K 2016 Dynamic multi-robot team reconfiguration using weighted voting games Journal of Experimental & Theoretical Artificial Intelligence 28 4 pp 607–628 doi: 10.1080/0952813X.2015.1020575.
[18] Efrima A and Peleg D 2009 Distributed algorithms for partitioning a swarm of autonomous mobile robots Theoretical Computer Science 410 14 pp 1355–1368 doi: 10.1016/j.tcs.2008.04.042.
[19] Défago X et al 2019 Fault-Tolerant Mobile Robots Distributed Computing by Mobile Entities (Theoretical Computer Science and General Issues 11340 1) eds Flocchini P et al Springer Nature Switzerland pp 234–251 doi: 10.1007/978-3-030-11072-7_10.
[20] Bhagat S et al 2019 Computation Under Restricted Visibility Distributed Computing by Mobile Entities (Theoretical
Computer Science and General Issues 11340 1) eds Flochini P et al Springer Nature Switzerland pp 134–183 doi: 10.1007/978-3-030-11072-7_7.

[21] Chatterjee A et al 2015 Gathering Asynchronous Swarm Robots under Non-uniform Limited Visibility Distributed Computing and Internet Technology (LNCS 8956) eds Natarajan R et al Springer Cham pp 174–180 doi: 10.1007/978-3-319-14977-6_11.

[22] Das D et al 2020 Assembling Multi-Robots Along a Boundary of a Region with Obstacles—A Performance Upgradation Advances in Computational Intelligence (AISC 988) eds Sahana S K et al Springer Singapore pp 201–12 doi: 10.1007/978-981-13-8222-2_17.

[23] Mohanty P K and Parhi D R 2016 Optimal path planning for a mobile robot using cuckoo search algorithm Journal of Experimental & Theoretical Artificial Intelligence 28 1–2 pp 35–52 doi:10.1080/0952813X.2016.1152789.

[24] Hassani I et al 2018 Robot Path Planning with Avoiding Obstacles in Known Environment Using Free Segments and Turning Points Algorithm Mathematical Problems in Engineering 2018 pp 1–13 doi:10.1155/2018/2163278.

[25] Zhu Z et al 2015 Global path planning of mobile robots using a memetic algorithm International Journal of Systems Science 46 11 pp 1982–1993 doi: 10.1080/00207721.2013.843735.

[26] Lee J H et al 2013 An Approach to Self-Assembling Swarm Robots Using Multitree Genetic Programming The Scientific World Journal 2013 pp 1–10 doi: 10.1155/2013/593848.

[27] Das D et al 2017 Multi-robot Assembling Along a Boundary of a Given Region in Presence of Opaque Line Obstacles Proceedings of 2nd International Conference on Intelligent Computing and Applications (AISC 467) eds Sundari P D et al Springer Singapore pp 21–29 doi:10.1007/978-981-10-1645-5_3.

[28] Tokas P et al 2017 Assembling Swarm with Limited Visibility in Presence of Line Obstacles Advances in Computing and Data Sciences (CCIS 721) eds Singh M et al Springer Singapore pp 60–69 doi: 10.1007/978-981-10-5427-3_7.