Gathering a Euclidean Closed Chain of Robots in Linear Time

Jannik Castenow(✉), Jonas Harbig, Daniel Jung, Till Knollmann, and Friedhelm Meyer auf der Heide

Heinz Nixdorf Institute and Computer Science Department, Paderborn University, Paderborn, Germany
{janniksu,jharbig,jungd,tillk,fmadh}@mail.upb.de

Abstract. We focus on the following question about Gathering of n autonomous, mobile robots in the Euclidean plane: Is it possible to solve Gathering of robots that do not agree on their coordinate systems (disoriented) and see other robots only up to a constant distance (limited visibility) in linear time? Up to now, such a result is only known for robots on a two-dimensional grid [1,8]. We answer the question positively for robots that are connected in one closed chain (like [1]), i.e., every robot is connected to exactly two other robots, and the connections form a cycle. We show that these robots can be gathered by asynchronous robots ($\mathcal{A}$sync) in $\Theta(n)$ epochs assuming the $\mathcal{LUMI}$ model [12] that equips the robots with locally visible lights like in [1,8]. The lights are used to initiate and perform so-called runs along the chain, which are essential for the linear runtime. Starting of runs is done by determining locally unique robots (based on geometric shapes of neighborhoods). In contrast to the grid [1], this is not possible in every configuration in the Euclidean plane. Based on the theory of isogonal polygons by Grünbaum [18], we identify the class of isogonal configurations in which, due to a high symmetry, no locally unique robots can be identified. Our solution consists of two algorithms that might be executed in parallel: The first one gathers isogonal configurations without any lights. The second one works for non-isogonal configurations; it is based on the concept of runs using a constant number of lights.

1 Introduction

The Gathering problem is one of the most studied and fundamental problems in the research area of distributed computing by mobile robots. Gathering requires a set of initially scattered point-shaped robots to meet at the same (not predefined) position. This problem has been studied under several different robot and time models, all having in common that the capabilities of the individual robots are very restricted. The central questions among all

This work was partially supported by the German Research Foundation (DFG) under the project number 453112019; ME 872/14-1. A brief announcement about the result under the $\mathcal{F}$sync scheduler has appeared in [5] and a full version can be found online [6].

L. Gąsieniec et al. (Eds.): ALGOSENSORS 2021, LNCS 12961, pp. 29–44, 2021.
https://doi.org/10.1007/978-3-030-89240-1_3

models are: Which capabilities of robots are needed to solve Gathering and how do these capabilities influence the runtime? One popular and well-studied model is the $\mathcal{OBLOT}$ model [15]. Its fundamental features are that the robots are *autonomous*, *identical* and *anonymous* (externally and internally identical), *homogeneous* (all robots execute the same algorithm), *silent* (no direct communication) and *oblivious* (no persistent memory). Additionally, the robots operate in discrete `Look-Compute-Move` cycles (rounds). Each cycle consists of three phases: first, robots take snapshots of their environment; afterward, they compute a target point and finally move there. The cycles of the robots can either be fully synchronous ($\mathcal{F}$sync), semi-synchronous ($\mathcal{S}$sync) or completely asynchronous ($\mathcal{A}$sync). Time is measured in *epochs*, i.e., the smallest number of rounds such that each robot has completed its cycle at least once. Another emerging model is the $\mathcal{LUMI}$ model [12,16] – it coincides in most parts with the $\mathcal{OBLOT}$ model. However, it does not demand the robots to be oblivious and silent. Instead, the robots are equipped with locally visible lights that are persistent. These lights can be used to communicate state information to local neighbors.

While it is nowadays well understood under which capabilities Gathering is possible, much less is known concerning how the capabilities influence the runtime. The best known algorithm in the Euclidean plane considering disoriented robots with limited visibility and the $\mathcal{OBLOT}$ model is the Go-To-The-Center (GTC) algorithm that requires $\Theta(n^2)$ rounds under the $\mathcal{F}$sync scheduler [11]. It is conjectured that the runtime is optimal for the given robot model. The $\Omega(n^2)$ lower bound of GTC examines an initial configuration where the robots form a regular polygon with neighboring robots having a constant distance, the *viewing radius*. It is shown that GTC takes $\Omega(n^2)$ rounds until the robots start seeing more robots than their initial neighbors. This gives rise to a slightly different connectivity model, the *closed chain* [1]. Each robot has two direct chain neighbors it can distinguish, and the chain connections form a cycle in a closed chain. A robot can always see a constant number of chain neighbors in each direction along the chain. Based on the observations above, it can be seen that GTC has a runtime of $\Theta(n^2)$ for closed chains. Hence, it is also still open whether closed chains of disoriented robots with limited visibility can be gathered in linear time in the Euclidean plane. For robots that are located on a two-dimensional grid, the picture is different: a linear time algorithm for closed chains exists [1]. The algorithm is based on (at least) two main concepts: the distinction of *connectivity range* and *viewing range* and a locally sequential movement called *run* implemented with the help of the $\mathcal{LUMI}$ model. It is assumed that the distance between two direct neighbors in the chain is at most 1 (the connectivity range), but the robots can see the next 11 (the viewing range) robots in each direction along the chain. The larger viewing range significantly enhances the local views of the robots and is a commonly used tool in the context of efficient Gathering algorithms, see, e.g., [1,8,24].

The second central concept behind the grid algorithm is the notion of a run (initially introduced in [22]). A run is a visible state (realized with lights) that is swapped along the chain and allows the robot with the state to move. In most rounds, $\Omega(n)$ runs are active, essential for the linear runtime. To start runs, locally unique robots – based on geometric shapes of neighborhoods – are determined. Therefore, the grid algorithm [1] benefits from an ever-present asymmetry of non-final configurations on the grid such that robots can always be identified to start runs (the "most symmetric" configuration w.r.t. the local views of the robots is the square). In the Euclidean plane, the picture is fundamentally different since classes of (non-final) configurations exist where every robot has the same view. The most obvious example is a configuration in which robots are located on the vertices of a regular polygon.

We show that closed chains of disoriented robots with limited visibility in the Euclidean plane can be gathered in linear time assuming the $\mathcal{LUMI}$ model. Our solution combines two algorithms: the first algorithm uses the notion of a run and identifies locally unique robots to start runs. The movement operations of robots with runs are inspired by [22], an algorithm for shortening open chains of robots in the Euclidean plane. In our model, and contrast to [22], runs might have different movement directions along the chain. The different directions introduce the additional challenge to handle runs with opposite movement directions located at direct neighbors. Another new challenge (compared to [22]) is to prevent runs from cycling multiple times around the chain. The second algorithm considers all configurations for which the concept of runs is not applicable due to the absence of locally unique robots. Based on the theory of *isogonal polygons* by Grünbaum [18], we characterize isogonal configurations in which no locally unique robots exist. With the help of the characterization, we introduce an algorithm that gathers isogonal configurations in linear time without using any light. We then demonstrate that running both algorithms in parallel, i.e., robots whose neighborhood fulfills the criterion of being an isogonal configuration, execute the algorithm for isogonal configurations while all others execute the asymmetric algorithm, solves Gathering in linear time.

Related Work: Much research is devoted to Gathering in various settings, mostly combined with an unbounded viewing radius. Due to space constraints, we focus on results that deal with the runtime of Gathering algorithms. For a comprehensive overview of models, algorithms, and analyses, we refer the reader to the recent survey [14]. Additionally, see [13,17,23,26] for practical applications of robot chains and [1,7,10,20,22,25,27] for more algorithmic results about robot chain problems.

In the $\mathcal{OBLOT}$ model, there is the GTC algorithm [2] that solves Gathering of disoriented robots with local visibility in $\Theta\left(n^2\right)$ rounds assuming the $\mathcal{F}$sync scheduler [11]. The same runtime can be achieved for robots located on a two-dimensional grid [4]. It is conjectured that both algorithms are asymptotically optimal and thus, $\Omega\left(n^2\right)$ is also a lower bound for *any* algorithm that solves Gathering in this model. The only proven non-trivial lower bound is $\Omega(D_G^2)$, where D_G denotes the diameter of the initial visibility graph [19]. However, this bound only holds for comparably small diameters $D_G \in \Theta(\sqrt{\log\ n})$. Faster runtimes could only be achieved in a continuous time model (see [21] for an overview), or by assuming agreement on one or two axes of the local coordinate systems or considering the $\mathcal{LUMI}$ model. In [24], an algorithm with runtime $\Theta(D_E)$ ($\mathcal{A}$SYNC) for robots in the Euclidean plane assuming one-axis agreement in the $\mathcal{OBLOT}$ model is introduced. D_E denotes the initial configuration's Euclidean diameter (the largest distance between any pair of robots). Assuming disoriented robots, the algorithms that achieve a runtime of $o(n^2)$ are developed under the $\mathcal{LUMI}$ model and assume robots that are located on a two-dimensional grid: There exist two algorithms having an asymptotically optimal runtime of $\mathcal{O}(n)$; one algorithm for *closed chains* [1] and another one for arbitrary (connected) swarms [8].

Our Contribution: In this work, we give the first asymptotically optimal algorithm, called CCH, that solves GATHERING of disoriented robots in the Euclidean plane. More precisely, we show that a closed chain of disoriented robots with limited visibility located in the Euclidean plane can be gathered in $\mathcal{O}(n)$ epochs assuming the $\mathcal{LUMI}$ model with a constant number of lights and the $\mathcal{A}$SYNC scheduler. The number of epochs is asymptotically optimal since if the initial configuration forms a straight line with direct neighbors at a maximal distance, at least $\Omega(n)$ epochs are required by *any* algorithm.

Our algorithm assumes that direct chain neighbors are in distance at most 1 (the connectivity range is 1), and robots can always see the positions of 4 robots in each direction along the chain (the viewing range is 4). The viewing range of 4 is a significant improvement over the linear time GATHERING algorithm for closed chains of robots on a grid that uses a viewing range of 11 [1].

The visible lights help to exploit asymmetries in the chain to identify locally unique robots that generate *runs*. We characterize the class of *isogonal configurations* based on the theory of isogonal polygons by Grünbaum [18] and show that no locally unique robots exist in these configurations, in contrast to every other configuration. We believe that this characterization is of independent interest as highly symmetric configurations often cause a large runtime. For instance the lower bound of GTC holds for an isogonal configuration [11].

Our approach combines two algorithms into one: An algorithm inspired by [1,22] that gathers non-isogonal configurations in linear time using visible lights and another algorithm for isogonal configurations without using any lights. Note that there might be cases in which both algorithms are executed in parallel due to the limited visibility. An additional rule ensures that both algorithms can be interleaved without hindering each other.

In this version of the paper, we introduce CCH for the $\mathcal{F}$SYNC scheduler. Due to space constraints, the two-step synchronization procedure (mainly based on existing results [3,9]) to make the algorithm work under the $\mathcal{A}$SYNC scheduler while maintaining the runtime of $\Theta(n)$ epochs can be found in the full version [6].

2 Model and Notation

Time Model: Robots operate in discrete LCM (`Look`, `Compute`, `Move`) cycles. Each robot takes a snapshot of its neighborhood during `Look`, computes a target point in `Compute`, and moves to this point in `Move`. We assume a *rigid* movement, robots always reach their target points during `Move`. A scheduler controls the timing of the executions of the LCM cycles: The cycles can be fully synchronous ($\mathcal{F}$SYNC), or only a subset of all robots participate ($\mathcal{S}$SYNC). Additionally, the cycles can be asynchronous ($\mathcal{A}$SYNC). The executions of the $\mathcal{S}$SYNC and $\mathcal{A}$SYNC schedulers are always fair: All robots execute their cycles infinitely often. The $\mathcal{S}$SYNC and $\mathcal{A}$SYNC schedulers are called to be *k-fair* if, between any two successive cycles of a robot, every other robot is activated at most k times. Time is measured in epochs, i.e., the smallest number of rounds until each robot processed at least one complete LCM cycle.

Robot Model: We consider n robots $r_0, \ldots, r_{n-1}$ located in $\mathbb{R}^2$. Each robot occupies a single point, and there can be multiple robots in the same location. Moreover, the robots are connected in a closed chain topology: Each robot r_i has two direct neighbors: r_{i-1} and r_{i+1} (mod n). The *connectivity range* is assumed to be 1, i.e., two direct neighbors are allowed to have a distance of at most 1. The robots are disoriented and thus do not agree on any axis of their local coordinate systems, and the latter can be arbitrarily rotated and inverted. However, the robots agree on unit distance and can measure distances precisely. Except for their direct neighbors, each robot can see the positions of 4 predecessors and successors along the chain. Moreover, we assume the $\mathcal{LUMI}$ model: Each robot is equipped with a constant number of lights $\ell_1, \ldots, \ell_k$ with color sets $C_1, \ldots, C_k$ and at every point in time each light can have a single color out of its color set (later on, we use names like ℓ_r).[1]

Notation: Let $p_i(t)$ be the position of r_i in round t in a global coordinate system (not known to the robots) and $d(p_i(t), p_j(t)) = \|p_i(t) - p_j(t)\|$. Furthermore, let $u_i(t) = p_i(t) - p_{i-1}(t)$ be the vector pointing from robot r_{i-1} to r_i in round t. The length of the chain is defined as $L(t) := \sum_{i=0}^{n-1} \|u_i(t)\|$. The angle created by anchoring $u_{i+1}(t)$ at the terminal point of $u_i(t)$ is denoted by $\alpha_i(t) = \measuredangle(u_i(t), u_{i+1}(t)) \in [0, \pi]$, $\mathrm{sgn}_i\left(\alpha_i(t)\right) \in \{-1, 0, 1\}$ denotes the orientation of $\alpha_i(t)$ from r_i's point of view and $\mathrm{sgn}(\alpha_i(t))$ denotes the orientation in a global coordinate system. $N_i(t)$ denotes the neighborhood of a robot. Throughout the algorithm's execution, two robots may merge and continue to behave as a single robot. For simplicity, r_{i+1} represents the first robot with an index larger i that has not yet merged with r_i (r_{i-1} analogously).

3 Basics

This section explains the basics behind the two sub-algorithms that are part of the CCH. In Sect. 3.1 we characterize isogonal configurations. Section 3.2 explains the basic idea of the asymmetric algorithm: the notion of runs and the movement operations induced by them. The $\mathcal{F}$SYNC scheduler is considered.

[1] In the classical $\mathcal{LUMI}$ model [12] each robot is equipped with a single light and color set. Our assumption of multiple lights and color sets can be transferred to the classical setting by choosing a single light with a color set of size at most $2^{\sum_{i=1}^{k} |C_i|}$.

3.1 Isogonal Configurations

There are some configurations in which every robot locally has the same view. We classify these as the *isogonal configurations.* The most prominent example is a configuration where robots are located on the vertices of a regular polygon. However, there are more such configurations. Intuitively, a configuration is isogonal if all angles $\alpha_i(t)$ have the same size and orientation and either all vectors $u_i(t)$ have the same length, or there are two alternating vector lengths. More formally, for some round t, the set of all vectors $u_i(t)$ describes a polygon denoted as the *configuration polygon* of round t. A configuration is then called an *isogonal configuration* in case its configuration polygon is isogonal. Grünbaum classified the isogonal polygons as follows [18]: A polygon P is *isogonal* iff for each pair of vertices there is a symmetry of P that maps the first onto the second [18]. Examples of such polygons can be seen in Figs. 1 and 2. The set of isogonal polygons consists of the *regular stars* and polygons that can be obtained from them by a small translation of the vertices [18]. A *regular star* $\{n/d\}$ $(n, d \in \mathbb{N}, d \leq n)$ is constructed as follows: Consider a circle C and fix an arbitrary radius R of C. Place n points $A_1, \ldots, A_n$ such that A_j is placed on C and forms an angle of $2\pi d/n \cdot j$ with R and connect A_j to $A_{j+1 \bmod n}$ by a segment. A configuration is called a *regular star configuration*, in case the configuration polygon is a regular star [18].

For odd n, every isogonal polygon is a regular star. For even n, isogonal polygons that are not regular stars can be constructed as follows: Take any regular star $\{n/d\}$ based on the circle C of radius R. Choose a parameter $0 < t < n/2$ and locate the vertex A_j such that its angle to R is $2\pi/n \cdot (j \cdot d + (-1)^j \cdot t)$. Choosing $t = \frac{n}{2}$ yields the polygon $\{n/d\}$ again. Larger values for t obtain the same polygons as in the interval $[0, \frac{n}{2}]$ [18].

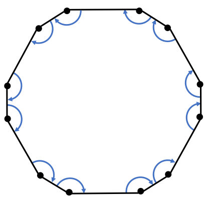
Fig. 1. An isogonal configuration that has two alternating vector lengths and all angles are equal with $n = 12$.

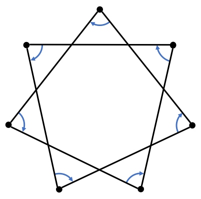
Fig. 2. An isogonal configuration of which the polygon is a star configuration with $n = 7$.

3.2 Sequential Movement with Run-States

A run-state (introduced first in [22]) is a light that is passed along the chain in a fixed direction associated to it. Robots with a run-state perform a movement operation while robots without do not. The movement is sequentialized in a way that in round t the robot r_i executes a movement operation (and neither r_{i-1} nor r_{i+1}), the robot r_{i+1} in round $t+1$ and so on (cf. Fig. 3). The movement of a run-state along the chain is denoted as a *run*. For ease of description, we use the term *run* (instead of run-state) for the rest of the paper. Due to the sequential passing of runs, any moving robot does not have to consider the movements of its direct neighbors since it knows that these do not change their positions. There is one exception: two runs with opposite directions might be located at two neighboring robots. In this case, the two robots move simultaneously. A run can be implemented with two lights: ℓ_r and ℓ_p. The light ℓ_r indicates that a robot has a run in the current round, and ℓ_p is active if a robot had a run in the last round. Thus, each run keeps a fixed direction along the chain; robots that have not activated the light ℓ_p and see one neighbor with an active light ℓ_r will take over the run in the next round by activating ℓ_r. After completing the movement based on the run, ℓ_r is switched off, and ℓ_p is activated such that the robot does not take over the same run in the next round. We use the following notation to speak about runs. For a robot r_i, $run(r_i, t) = true$ if r_i has a run in round t. Additionally, $run(N_i(t)) = \{r_j \in N_i(t) \mid run(r_j, t) = true\}$. Let κ denote an arbitrary run. $r(\kappa, t)$ denotes the robot that has run κ in round t.

Movement Operations Based on Runs: To preserve the connectivity of the chain, CCH ensures that at most two directly neighboring robots move in the same round. This is done by allowing the existence of only two patterns of runs at neighboring robots: Either r_i and neither r_{i-1} nor r_{i+1} has a run (*isolated run*) or r_i and r_{i+1} have runs heading in each other's direction while r_{i-1} and r_{i+2} do not have runs (*joint run-pair*). All other patterns, especially sequences of length at least 3 of neighboring robots having runs, are prohibited.

For robots with a run, there are three kinds of movement operations, the *merge*, the *shorten* and the *hop*. The purpose of the merge is to reduce the number of robots in the chain. It is executed by a robot r_i if its direct neighbors have a distance of at most 1. In this case, r_i is not necessary for the connectivity of the chain and can be safely removed. Removing r_i means that it moves to the position of its next neighbor in the direction of the run, the robots merge their neighborhoods, and both continue to behave as a single robot. The execution of a merge stops a run. The goal of a shorten is to reduce $L(t)$ by moving to the midpoint between the direct neighbors. After executing a shorten, the run stops. In case no significant progress can be made locally, a hop is executed. The purpose of a hop is to exchange two neighboring vectors in the chain. By this, each run is associated with a run-vector. The vector is swapped along the chain until it finds a position at which a merge or a shorten can be executed. For each of the three operations, there is also a *joint* one (*joint hop*, *joint shorten* and *joint merge*), which is a similar operation executed by a joint run-pair. We continue with introducing the formal definitions of all movement operations. For the ease of notation, we assume for an isolated run κ that $r(\kappa, t) = r_i$ and $r(\kappa, t+1) = r_{i+1}$.

(Joint) Hop: Consider the isolated run κ. If r_i executes a *hop*, $p_i(t+1) = p_{i+1}(t) - u_i(t)$. The run continues in its direction. A *joint hop* is a similar operation executed by a joint run-pair κ_1 and κ_2 located at robots r_i and r_{i+1}. The new positions are $p_i(t+1) = p_{i-1}(t)+u_{i+2}(t)$ and $p_{i+1}(t+1) = p_{i+2}(t)-u_i(t)$. Both runs continue in their directions and skip the next robot, i.e., in round $t+1$, $r(\kappa_1, t+1) = r_{i+2}$ and $r(\kappa_2, t+1) = r_{i-1}$. See Fig. 4 for a visualization.

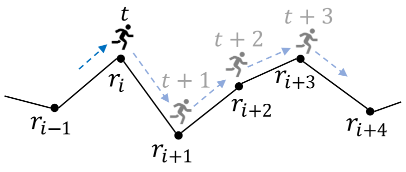

Fig. 3. A run at r_i in round t is passed along the chain.

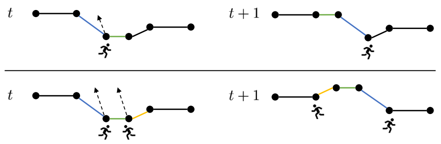

Fig. 4. Visualization of a hop (above) and a joint hop (below).

(Joint) Shorten: In the *shorten*, a robot r_i with an isolated run moves to the midpoint between its direct neighbors: $p_i(t+1) = \frac{1}{2} \cdot p_{i-1}(t) + \frac{1}{2} \cdot p_{i+1}(t)$. The run stops. In a *joint shorten* executed by two robots r_i and r_{i+1} with a joint run-pair, the vector $v(t) = p_{i+2}(t) - p_{i-1}(t)$ is subdivided into three parts of equal length. The new positions are $p_i(t+1) = p_{i-1}(t) + \frac{1}{3} \cdot v(t)$ and $p_{i+1}(t+1) = p_{i+2}(t) - \frac{1}{3} \cdot v(t)$. Both runs are stopped after executing a joint shorten. See Fig. 5 for a visualization of both operations.

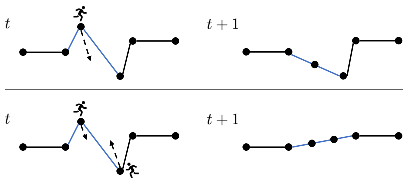

Fig. 5. Visualization of a shorten (above) and a joint shorten (below).

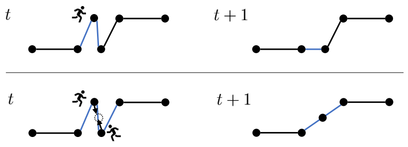

Fig. 6. Visualization of a merge (above) and a joint merge (below).

(Joint) Merge: If r_i executes a *merge*, it moves to $p_{i+1}(t)$. Afterward, the robots r_i and r_{i+1} merge such that their neighborhoods are identical, and they continue to behave like a single robot. In the *joint merge*, the robots r_i and r_{i+1} both move to $\frac{1}{2}p_i(t) + \frac{1}{2}p_{i+1}(t)$. The robots merge there such that they behave as a single robot in the future. All runs that participate in a (joint) merge are immediately stopped (Fig. 6).

4 Closed-Chain-Hopper

Next, we present the Closed-Chain-Hopper (CCH) algorithm under the $\mathcal{F}$sync scheduler in detail. Our approach consists of two algorithms – one for asymmetric configurations (Sect. 4.2) and one for isogonal configurations (Sect. 4.3). All robots with an isogonal neighborhood move according to the symmetric algorithm. Other robots follow the asymmetric algorithm. To present the algorithm in the most comprehensible way, we use an incremental description. For the sake of completeness, a pseudocode can be found in the full version [6].

4.1 Intuition About the Asymmetric Algorithm

While Sect. 3.2 has already discussed the purpose of the individual movement operations, we add some intuition about how to handle runs in general.

Generation of Runs: In non-isogonal configurations, we identify robots that are regarding their local neighborhood geometrically unique. These robots are assigned an init-state (implemented with a light ℓ_{init}) allowing them to regularly generate new runs. The algorithm always ensures that at most two neighboring robots have an init-state to maintain the connectivity of the chain. Additionally, a robot r_i with an init-state only generates a new run in case no other run is present in its neighborhood.

Stopping of Runs After at Most n Rounds: For the linear runtime of the CCH algorithm, each run must stop after at most n rounds since otherwise the run could cycle multiple times around the chain and hinder other robots with init-states to generate new runs that potentially lead to progress. The following ideas are used to ensure this behavior. Robots with init-states generate two runs at the same time: one run heading in each direction of the chain. Additionally, the robot with the init-state moves to the midpoint between its two direct neighbors before generating the runs. As a consequence, both runs start with opposite run-vectors. Furthermore, a hop is only executed if the angle between two neighboring vectors is larger than $7/8\pi$. Now suppose that the robot starting the two runs lies in the origin of a global coordinate system, and after moving to the midpoint, one of its neighbors lies on the positive x-axis while the other one lies on the negative x-axis. The angle of $7/8\pi$ ensures that the run that starts along the positive x-axis can only move to the right in case of a (joint) hop while the other run can only move to the left. Hence, the two runs cannot meet each other again, and at least one run must stop via a (joint) merge or a (joint) shorten. See Fig. 7 for a visualization. Observe that a threshold of $\frac{\pi}{2}$ would be sufficient to ensure that only one run stops.

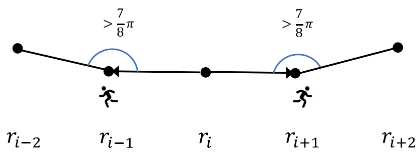

Fig. 7. Two runs generated by r_i with opposite run-vectors. Due to the threshold of $7/8\pi$, the run at r_{i-1} can only execute a hop if r_{i-2} is positioned to the left of r_{i-1} (mirrored for r_{i+1}).

A larger threshold is needed to guarantee that the second run stops after at most n rounds. Suppose the first run (w.l.o.g. the run that moves to the left) stops at the robot r_i. The algorithm needs to ensure that r_i cannot move to the right of the second run to ensure that the second run also stops. For this, the threshold of $7/8\pi$ is crucial as well as an additional rule to ensure that the chain structure cannot change significantly because of (joint) merges. The rule is defined as follows: Each run that leads to a (joint) merge stops all runs in its neighborhood. Additionally, for a constant number of rounds also no new run is generated in this neighborhood. This way, it is ensured that not too many (joint) merges occur during n rounds to guarantee also the second run to stop.

4.2 Asymmetric Algorithm in Detail

The asymmetric algorithm consists of two parts: The generation of new runs and the movement depending on such a run. We start by explaining the movement depending on runs. Assume that the number of robots in the chain is at least 6 and consider an isolated run κ in round t with $r(\kappa, t) = r_i$ and $r(\kappa, t+1) = r_{i+1}$. Then, r_i moves as described in Fig. 8. Given a joint run-pair at robots r_i and r_{i+1}, the robots r_i and r_{i+1} move according to Fig. 9. If the number of robots in the chain is at most 5 (robots can detect this since they can see 4 robots in each direction), the robots move towards the center of the smallest enclosing circle of their neighborhood while ensuring connectivity. More precisely, the robots execute GTC which ensures Gathering after $\mathcal{O}(1)$ rounds [11].

1. If $d(p_{i-1}(t), p_{i+1}(t)) \leq 1$,	r_i: merge.
2. If $d(p_i(t), p_{i+2}(t)) \leq 1$,	r_i: Pass the run to r_{i+1}.
3. If $\alpha_i(t) \leq 7/8\pi$,	r_i: shorten.
4. Otherwise,	r_i: hop.

Fig. 8. Movement of isolated runs.

1. If $d(p_{i-1}(t), p_{i+2}(t)) < 2$,	both: joint merge.
2. If $\alpha_i(t) \leq 7/8\pi$ and $\alpha_{i+1}(t) \leq 7/8\pi$	both: joint shorten
3. If $\alpha_i(t) \leq 7/8\pi$,	r_i: shorten.
4. If $\alpha_{i+1}(t) \leq 7/8\pi$,	r_{i+1}: shorten.
5. If $\measuredangle(u_i(t), -u_{i+2}(t)) \leq 7/8\pi$.	both: joint shorten.
6. Otherwise,	both: joint hop.

Fig. 9. Movement of joint run-pairs.

Where to Start Runs? New runs are created by robots with init-states (realized with a light ℓ_{init}). We say $init(r_i) = true$ if r_i has an init-state. Moreover, $init(N_i(t)) = \{r_j \in N_i(t) \,|\, init(r_j) = true\}$. To generate new init-states, we aim at discovering structures in the chain that are asymmetric. When the surrounding robots observe such a structure, the robot closest to the structure is assigned an init-state. Our rules ensure that at most two neighboring robots have an init-state to keep the distance between runs (essential for maintaining the connectivity). A robot r_i only tries to assign itself a init-state if $init(N_i(t)) = \emptyset$. There are three sources of asymmetry in the chain: Sizes of angles, orientations of angles, and lengths of vectors. For each source of asymmetry, we introduce a set of patterns. The next class of patterns is only checked if a complete symmetry regarding the previous pattern is identified. More precisely, a robot only checks orientation patterns if all angles $\alpha_i(t)$ in its neighborhood are identical. Similarly, a robot only checks vector length patterns if all angles in its neighborhood have the same size and orientation. Whenever a pattern holds, the robot observing the pattern assigns itself an init-state if there is no other robot already assigned an init-state in its neighborhood. If two direct neighbors are assigned an init-state, they fulfill the same type of pattern and form a *joint init-state* together. For better readability, we omit the time parameter t, e.g., we write u_i instead of $u_i(t)$.

Angle Patterns: A robot r_i is assigned an init-state if either $\alpha_{i-1}(t) > \alpha_i(t) \leq \alpha_{i+1}(t)$ or $\alpha_{i-1}(t) \geq \alpha_i(t) < \alpha_{i+1}(t)$ (Fig. 10).

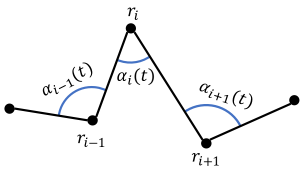

Fig. 10. A configuration in which r_i fulfills the first *Angle Pattern*, i.e., $\alpha_i(t)$ is a local minimum.

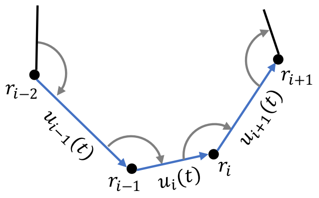

Fig. 11. A configuration in which r_i (and also r_{i-1}) fulfills the first *Vector Length Pattern*, i.e., the length of $u_i(t)$ is a local minimum.

Orientation Patterns: A robot r_i gets an init-state if one of the following patterns is fulfilled.

1. r_i is between three angles that have a different orientation than $\alpha_i(t)$: $\text{sgn}_i(\alpha_{i-1}(t)) = \text{sgn}_i(\alpha_{i+1}(t)) = \text{sgn}_i(\alpha_{i+2}(t)) \neq \text{sgn}_i(\alpha_i(t))$ or $\text{sgn}_i(\alpha_{i-2}(t)) = \text{sgn}_i(\alpha_{i-1}(t)) = \text{sgn}_i(\alpha_{i+1}(t)) \neq \text{sgn}_i(\alpha_i(t))$ (Fig. 12).
2. r_i borders a sequence of at least two angles with the same orientation next to a sequence of at least three angles with the same orientation: $\text{sgn}_i(\alpha_{i-1}(t)) = \text{sgn}_i(\alpha_i(t)) \neq \text{sgn}_i(\alpha_{i+1}(t)) = \text{sgn}_i(\alpha_{i+2}(t)) = \text{sgn}_i(\alpha_{i+3}(t))$ or $\text{sgn}_i(\alpha_{i+1}(t)) = \text{sgn}_i(\alpha_i(t)) \neq \text{sgn}_i(\alpha_{i-1}(t)) = \text{sgn}_i(\alpha_{i-2}(t)) = \text{sgn}_i(\alpha_{i-3}(t))$ (Fig. 13).[2]

[2] Observe that the viewing range of 4 is based on this pattern: to identify the angles $\alpha_{i+3}(t)$ and $\alpha_{i-3}(t)$, r_i needs to be able to see r_{i-4} and r_{i+4}.

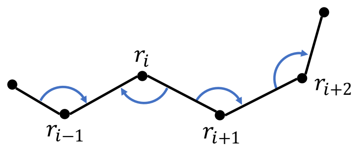

Fig. 12. A configuration in which r_i fulfills the first *Orientation Pattern.*

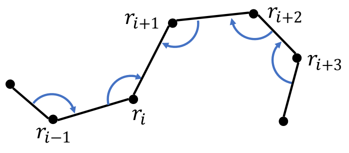

Fig. 13. A configuration in which r_i fulfills the second *Orientation Pattern.*

Vector Length Patterns: In the patterns, the term *locally minimal* occurs. $\|u_i\|$ is locally minimal means that all other vectors that can be seen by r_i are either larger or have the same length.

1. The robot is located at a locally minimal vector next to two succeeding larger vectors, i.e., $\|u_i\|$ is locally minimal **and** $\|u_{i-1}\| > \|u_i\| < \|u_{i+1}\|$ **and** $\|u_i\| < \|u_{i+2}\|$ **or** $\|u_{i+1}\|$ is locally minimal **and** $\|u_i\| > \|u_{i+1}\| < \|u_{i+2}\|$ **and** $\|u_{i+1}\| < \|u_{i+3}\|$ (Fig. 11).
2. The robot is at the boundary of a sequence of at least two locally minimal vectors, i.e., $\|u_{i-1}\| = \|u_i\| < \|u_{i+1}\|$ **or** $\|u_i\| > \|u_{i+1}\| = \|u_{i+2}\|$.

How to Start Runs? Robots with (joint) init-states try every 9 rounds (counted with a light ℓ_c) to start new runs. A robot r_i with $init(r_i) = true$ only starts a new run if $run(N_i(t)) = \emptyset$ to ensure sufficient distance between runs. The constant 9 is chosen to ensure that a robot (potentially) observes different runs in its neighborhoods each time it tries to start a new run. Additionally, r_i only starts new runs provided $d(p_{i-1}(t), p_{i+1}(t)) > 1$. Otherwise, it executes a merge. Given $d(p_{i-1}(t), p_{i+1}(t)) > 1$, r_i generates two new runs at its direct neighbors with opposite directions as follows: r_i executes a shorten and generates two new runs κ_1 and κ_2 with $r(\kappa_1, t+1) = r_{i+1}$ and $r(\kappa_2, t+1) = r_{i-1}$. Two robots r_i and r_{i+1} with a joint init-state proceed similarly: given $d(p_{i-1}(t), p_{i+2}(t)) \leq 2$, they directly execute a joint merge. Otherwise r_i and r_{i+1} execute a joint shorten and induce two new runs at their direct neighbors with opposite direction.

Blocking of Robots After (Joint) Merges: Suppose the robot r_i (and r_{i+1}) executes a merge (a joint merge). All runs in the neighborhood of r_i (and r_{i+1}) are immediately stopped and all robots in $N_i(t)$ do not start any further runs within the next 4 rounds (the robots are *blocked*, counted with a light ℓ_{block}). Special care has to be taken of init-states. Suppose that a robot r_i executes a merge into the direction of r_{i+1} while having an init-state. The init-state is handled as follows: In case $init(r_{i+2}) = false$ and r_{i+2} does not execute a merge in the same round and $init(r_{i+3}) = true$, the init-state of r_i is passed to r_{i+1}. Otherwise, the state is removed.

4.3 Symmetric Algorithm

A robot r_i moves according to the symmetric algorithm if its neighborhood is isogonal, i.e., all $\alpha_i(t)$ in its neighborhood have the same size and orientation, either all vectors $u_i(t)$ have the same length or have two alternating lengths, $init(N_i(t)) = \emptyset$ and $run(N_i(t)) = \emptyset$. Then, r_i performs one of the two following symmetrical operations. In case all vectors $u_i(t)$ have the same length, it performs a *bisector-operation*. The purpose of the bisector-operation is to move all robots towards the center of the circle surrounding the isogonal polygon described by the configuration. Otherwise, the robot executes a *star-operation*. The goal of the star-operation is to transform an isogonal configuration with two alternating vector lengths into a regular star configuration such that bisector-operations are applied afterward. More formally, in the *bisector-operation*, a robot r_i computes the angle bisector of vectors pointing to its direct neighbors (bisecting the angle of size less than π) and jumps to the point p on the bisector such that $d(p_{i-1}(t), p) = d(p_{i+1}(t), p) = 1$. If $d(p_i(t), p) > \frac{1}{5}$, the robot moves only a distance of $\frac{1}{5}$ towards p. Additionally, the *star-operation* works as follows: Let C be the circle induced by r_i's neighborhood and R its radius. If the diameter of C has a length of at most 2, r_i jumps to the midpoint of C. Otherwise, the robot r_i observes the two circular arcs $L_\alpha = \alpha \cdot R$ and $L_\beta = \beta \cdot R$ connecting itself to its direct neighbors. The angles α and β are the corresponding central angles measured from the radius R_i connecting r_i to the midpoint of C. W.l.o.g. assume $L_\alpha < L_\beta$. r_i jumps to the point on L_β such that L_α is enlarged by $R \cdot ((\beta - \alpha)/4)$.

4.4 Combination of the Algorithms

Intuitively, suppose some robots follow the asymmetric algorithm while others execute the symmetric algorithm. In that case, there are borders at which a robot r_i executes the symmetric algorithm while its direct neighbor does not move at all (since r_i moves according to the symmetric algorithm, the direct neighbor cannot have a run). At these borders, it can happen that the length of the chain increases. To prevent this from happening too often, we make use of an additional visible light ℓ_{sy}. Robots that move according to the symmetric algorithm store this via activating ℓ_{sy}. If any robot detects in the next round that ℓ_{sy} is activated, but its local neighborhood does not fulfill the criterion of being an isogonal configuration, it concludes that the chain is not entirely isogonal. The robot r_c closest to the asymmetry is assigned an init-state to ensure that this does not occur again. As a consequence, r_c and all robots that can see r_c will not execute the symmetric algorithm again until r_c executes a (joint) merge. Hence, this case can occur at most n times (for more details, see [6]).

4.5 Analysis Sketch

This section contains the analysis outline for proving the main theorem (Theorem 1) of the CCH algorithm. The proofs can be found in [6].

Theorem 1. *The* CCH*-algorithm gathers a closed chain of disoriented robots with a connectivity range of* 1 *and viewing range of* 4 *in* $\Theta(n)$ *rounds.*

One of the crucial properties for the correctness of the CCH-algorithm is that it maintains the connectivity of the chain.

Lemma 1. *In every round t, the configuration is connected.*

The asymmetric algorithm depends on the generation of runs. We prove that in every asymmetric configuration, at least one pattern is fulfilled.

Lemma 2. *A configuration without any init-state in round t becomes either isogonal or at least one init-state exists in round $t+1$.*

The following is the key lemma of the asymmetric algorithm: Every run started at robot r_i will never visit r_i again in the future. See Sect. 4.1 for an intuition.

Lemma 3. *A run does not visit the same robot twice.*

Next, we count the number of required runs to gather all robots. There can be at most $n-1$ (joint) merges. To count the number of (joint) shortens, one can see that a (joint) shorten either decreases $L(t)$ by a constant or a vector of length less than $\frac{1}{2}$ has a length of at least $\frac{1}{2}$ afterward. Both cases occur at most a linear number of times.

Lemma 4. *At most* $143\,n$ *runs are required to gather all robots.*

Additionally, we prove that a sufficient number of runs is generated by applying a witness argument. Consider an init-state. After 9 rounds, this state either creates a new run or waits since a run is in its neighborhood. This way, we can count each 9 rounds a new run: Either the robot with the init-state starts a new run or waits because of a different run. Roughly said, we can prove that in k rounds $\approx \frac{k}{9}$ runs exist. This holds until the init-state is removed due to a (joint) merge. Afterward, we continue counting at the next init-state in the direction of the run causing the (joint) merge.

Lemma 5. *A configuration that does not become isogonal gathers in* $\mathcal{O}(n)$ *rounds.*

The first step of the symmetric algorithm transforms an isogonal configuration into a regular star configuration.

Lemma 6. *Given an isogonal configuration with two alternating vector lengths in round t, the configuration is a regular star configuration in round $t+1$.*

To prove a linear runtime for regular star configurations, we analyze the runtime for the regular polygon $\{n/1\}$. In all other regular star configurations, inner angles are smaller, and the robots can move farther towards the center of the surrounding circle.

Lemma 7. *Regular star configurations gather in at most* $30\,n$ *rounds.*

5 Concluding Remarks

For Gathering of disoriented robots with limited visibility, still, no non-trivial lower runtime bound for oblivious robots is known. A slight exception might be the bound of [19] (see related work), which, however, only holds for comparably small diameters. We conjecture that the linear runtime of the CCH is only possible due to the visible lights, and thus, quadratic lower bounds hold for the oblivious case. However, proving a general lower bound seems to be quite challenging and is left for future research. Additionally, it is still open whether the result can be transferred to robot swarms without a chain topology. The main idea would be to apply the CCH algorithm to the boundaries of the swarm, which form cycles. First attempts show that this approach has problems with maintaining the connectivity to *inner* robots that are not part of a boundary.

References

1. Abshoff, S., Cord-Landwehr, A., Fischer, M., Jung, D., Meyer auf der Heide, F.: Gathering a closed chain of robots on a grid. In: 2016 IEEE International Parallel and Distributed Processing Symposium, IPDPS 2016, Chicago, IL, USA, 23–27 May 2016, pp. 689–699 (2016)
2. Ando, H., Oasa, Y., Suzuki, I., Yamashita, M.: Distributed memoryless point convergence algorithm for mobile robots with limited visibility. IEEE Trans. Robot. Autom. **15**(5), 818–828 (1999)
3. Awerbuch, B.: Complexity of network synchronization. J. ACM **32**(4), 804–823 (1985)
4. Castenow, J., Fischer, M., Harbig, J., Jung, D., Meyer auf der Heide, F.: Gathering anonymous, oblivious robots on a grid. Theor. Comput. Sci. **815**, 289–309 (2020)
5. Castenow, J., Harbig, J., Jung, D., Knollmann, T., Meyer auf der Heide, F.: Brief announcement: gathering in linear time: a closed chain of disoriented and luminous robots with limited visibility. In: Devismes, S., Mittal, N. (eds.) SSS 2020. LNCS, vol. 12514, pp. 60–64. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-64348-5_5
6. Castenow, J., Harbig, J., Jung, D., Knollmann, T., Meyer auf der Heide, F.: Gathering a euclidean closed chain of robots in linear time. CoRR abs/2010.04424 (2021). https://arxiv.org/abs/2010.04424
7. Cohen, R., Peleg, D.: Local spreading algorithms for autonomous robot systems. Theor. Comput. Sci. **399**(1–2), 71–82 (2008)
8. Cord-Landwehr, A., Fischer, M., Jung, D., Meyer auf der Heide, F.: Asymptotically optimal gathering on a grid. In: Proceedings of the 28th ACM Symposium on Parallelism in Algorithms and Architectures, SPAA 2016, Asilomar State Beach/Pacific Grove, CA, USA, 11–13 July 2016, pp. 301–312 (2016)
9. Das, S., Flocchini, P., Prencipe, G., Santoro, N., Yamashita, M.: Autonomous mobile robots with lights. Theor. Comput. Sci. **609**, 171–184 (2016)
10. Degener, B., Kempkes, B., Kling, P., Meyer auf der Heide, F.: Linear and competitive strategies for continuous robot formation problems. ACM Trans. Parallel Comput. **2**(1), 2:1–2:18 (2015)

11. Degener, B., Kempkes, B., Langner, T., Meyer auf der Heide, F., Pietrzyk, P., Wattenhofer, R.: A tight runtime bound for synchronous gathering of autonomous robots with limited visibility. In: SPAA 2011: Proceedings of the 23rd Annual ACM Symposium on Parallelism in Algorithms and Architectures, San Jose, CA, USA, 4–6 June 2011, pp. 139–148. ACM (2011)
12. Di Luna, G., Viglietta, G.: Robots with lights. In: Distributed Computing by Mobile Entities, Current Research in Moving and Computing, pp. 252–277 (2019)
13. Dixon, C., Frew, E.W.: Maintaining optimal communication chains in robotic sensor networks using mobility control. Mob. Netw. Appl. **14**(3), 281–291 (2009)
14. Flocchini, P.: Gathering. In: Distributed Computing by Mobile Entities, Current Research in Moving and Computing, pp. 63–82 (2019)
15. Flocchini, P., Prencipe, G., Santoro, N.: Moving and computing models: robots. In: Distributed Computing by Mobile Entities, Current Research in Moving and Computing, pp. 3–14 (2019)
16. Flocchini, P., Santoro, N., Wada, K.: On memory, communication, and synchronous schedulers when moving and computing. In: OPODIS. LIPIcs, vol. 153, pp. 25:1–25:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2019)
17. Gao, Y., Chen, H., Li, Y., Lyu, C., Liu, Y.: Autonomous wi-fi relay placement with mobile robots. IEEE/ASME Trans. Mechatron. **22**(6), 2532–2542 (2017)
18. Grünbaum, B.: Metamorphoses of polygons. In: The Lighter Side of Mathematics, pp. 35–48 (1994)
19. Izumi, T., Kaino, D., Potop-Butucaru, M., Tixeuil, S.: On time complexity for connectivity-preserving scattering of mobile robots. Theor. Comput. Sci. **738**, 42–52 (2018)
20. Kling, P., Meyer auf der Heide, F.: Convergence of local communication chain strategies via linear transformations: or how to trade locality for speed. In: SPAA 2011: Proceedings of the 23rd Annual ACM Symposium on Parallelism in Algorithms and Architectures, San Jose, CA, USA, 4–6 June 2011, pp. 159–166 (2011)
21. Kling, P., Meyer auf der Heide, F.: Continuous protocols for swarm robotics. In: Flocchini, P., Prencipe, G., Santoro, N. (eds.) Distributed Computing by Mobile Entities. LNCS, vol. 11340, pp. 317–334. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-11072-7_13
22. Kutylowski, J., Meyer auf der Heide, F.: Optimal strategies for maintaining a chain of relays between an explorer and a base camp. Theor. Comput. Sci. **410**(36), 3391–3405 (2009)
23. Nguyen, H.G., Pezeshkian, N., Raymond, S.M., Gupta, A., Spector, J.M.: Autonomous communication relays for tactical robots. In: Proceedings of the 11th International Conference on Advanced Robotics (ICAR), pp. 35–40 (2003)
24. Poudel, P., Sharma, G.: Universally optimal gathering under limited visibility. In: Spirakis, P., Tsigas, P. (eds.) SSS 2017. LNCS, vol. 10616, pp. 323–340. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-69084-1_23
25. Regnault, D., Rémila, E.: Lost in self-stabilization: a local process that aligns connected cells. Theor. Comput. Sci. **736**, 41–61 (2018)
26. Tagliabue, A., Schneider, S., Pavone, M., Agha-mohammadi, A.: Shapeshifter: a multi-agent, multi-modal robotic platform for exploration of titan. CoRR abs/2002.00515 (2020). https://arxiv.org/abs/2002.00515
27. Tekdas, O., Plonski, P.A., Karnad, N., Isler, V.: Maintaining connectivity in environments with obstacles. In: IEEE International Conference on Robotics and Automation, ICRA 2010, Anchorage, Alaska, USA, 3–7 May 2010, pp. 1952–1957 (2010)