A Study of Mobile Robot Algorithms with Sycamore

by

Harish Prakash

Thesis submitted to the
Faculty of Graduate and Postdoctoral Studies
In partial fulfillment of the requirements
For the Master degree in
Computer Science.

School of Electrical Engineering and Computer Science
Faculty of Engineering
University of Ottawa

Abstract

In this thesis we considered a simulation platform for mobile robots algorithms: Sycamore. We implemented several new features for Sycamore and we tested them while studying three different algorithms to achieve gathering by robots with limited visibility: a deterministic well known algorithm, a simple new probabilistic algorithm, and a combination of the two.

The deterministic algorithm is known to achieve exact gathering when there are no faults; we tested it for the first time in presence of crashes and observed interesting and unexpected behaviors. We then performed extensive simulations with the probabilistic solution to identify the cause of an unexpected high rate of success, the simulations help us identify the relation between the rate of success and the initial configuration. Finally, we combined the two designing a hybrid solution. This work resulted in improvements of Sycamore, which can now be better employed to study mobile robots algorithms, as well as in empirical observations leading to new theoretical problems to be investigated.

Acknowledgements

My heart is humbled to see the work that has manifested in this thesis. I glorify the Lord Jesus Christ who empowers me in all things, even in this work, through many difficulties. To mention some of His blessings: I am blessed with the supervision of Professor Paola Flocchini, I am thankful for the numerous hours of review she has put in, her wise suggestions and her patience. I am blessed with the help provided by Giovanni Viglietta with Mathematica. I am blessed with the accurate and prompt guidance by Annik Dion who has been of great help throughout this program and I am blessed by the University of Ottawa for this opportunity. I dedicate this work to my parents, Prakash Jagannathan and Vijaya Prakash, who have always encouraged me and lifted me up even when I failed. Finally, with a fervent desire to serve this community of researchers, I present "A Study of Mobile Robot Algorithms with Sycamore"

Contents

List of Algorithms

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Background, Motivations, and Goals

1.1.1 Mobile Robots

Mobile robots are autonomous computational entities (modeled as points) that move on the plane. Their job is to perpetually repeat *look-compute-move* cycles, known as their life cycle. In the *Look* phase, a robot takes a snapshot of the visible portion of the environment recording the positions of the other robots; in the *Compute* phase, it calculates a destination point on the basis of the observed positions; in the *Move* phase, it moves to the computed destination.

Mobile robots can execute algorithms and perform tasks individually or as a group, and typically they take local decisions to solve some global problem. When working together, mobile robots can execute various tasks like gathering (i.e., moving to the same position, not decided a-priori), pattern formation (i.e., forming a given pattern), flocking (i.e., moving together in a given formation), scattering (i.e., homogeneously spreading in the environment).

A necessary ability to perform tasks in a group is communication; communication may be implicit or explicit. Explicit communication between mobile robots takes place

through a wireless network, flashing of lights or other signals. Implicit communication takes place when the robots infer information from the positions of the other robots they can see. In this thesis we only consider implicit communication.

Mobile robots may operate under a variety of different conditions (or assumptions), like visibility, scheduler, memory, agreement, multiplicity detection, and more. For example, the visibility could be *full* (in which case a robot can see all the robots in the universe every time it looks), or *limited* (in which case a robot can only see only robots located in a circle of a give radius around itself). The memory could be *unbounded*, in which case the mobile robot will remember every snapshot it ever saw and every destination it ever computed, or the robots could be *oblivious*, in which case the robots have absolutely no memory of their past cycles. The scheduler could range from *fully synchronous* (where all actions are synchronized) to *fully asynchronous* (where each robots is activated independently of the others).

In this thesis we consider only oblivious robots with limited visibility in fully synchronized environments. In other words, each robot can “see" only a circle around itself of a given radius, all robots execute their life cycles at the same time and, after executing a cycle a robot will forget all it did and saw in the previous cycle.

Oblivious mobile robot algorithms are an interesting field of research that has received a lot of attention lately (e.g., see [29]). One of the advantages of obliviousness is a certain level of fault-tolerance and self-stabilization. In fact, in presence of temporary faults, these robots will restore their task as soon as the faults leave the system, with no additional overhead. A fault is a mobile robot which does not perform its intended task; a crash is a special type of fault: a robot stops and will not move again.

One of the tasks mobile robots could perform together as a group is *Gathering*. Gathering algorithms cause mobile robots to move to a common position, and this is a typical necessary condition to be able to perform other more complex tasks which require the robots to start from the same place. Gathering has been studied quite extensively,

under many different conditions or assumptions, mostly in fault-free environments (e.g., see [15, 26, 32, 38, 46]). A few papers also consider gathering in presence of faults (e.g., see [6, 7, 8, 12]) by robots with full visibility.

Most investigations in this setting are done from a theoretical point of view. In particular, research effort has been made in the theoretical algorithmic community to design mobile robot algorithms and to understand the impact of various assumptions (e.g., agreement on a common coordinate system, possibility of distinguishing multiplicity of robots, visibility radius, etc.) on the feasibility of problems.

Moreover, the current work has focused almost exclusively on fault-free systems, where all the robots behave exactly as prescribed. Very little is known for situations where some robots are faulty and they do not obey to the algorithm in a precise way. This type of behaviors include imprecise vision, calculations and movements, Byzantine faults making the robots move in a random fashion, crash faults making them stop executing their cycle (as would happen if a battery expires or if the robot breaks). Theoretical study of swarm behaviors in presence of faults is extremely difficult and very little is known (e.g., [6, 7, 8]). In particular, nothing is known for the case when robots have limited visibility, which is much more difficult to treat theoretically.

1.1.2 Experimental Studies

It is clear that theoretical research would greatly benefit from a general purpose simulator which could perform experimental studies and could allow to better understand the robots' behaviors in a variety of different settings.

Perhaps surprisingly, experimental analysis in this context has not been developed very much. Some researchers built their own simulator to validate their algorithms also experimentally, but typical existing simulators are ad-hoc for specific settings or, even, for specific algorithms and cannot be adopted in different situations and under different assumptions on the robots' characteristics (e.g., see [14, 19]). To the best of our

knowledge, the only exception is Sycamore [52]. Sycamore has been recently developed as a versatile simulator which should be easily adaptable and customizable. It allows selection of properties and assumptions for the robots, and it is designed in such a way that programming specific algorithms should not be too complicated. Sycamore has been used almost exclusively to visualize some specific algorithms, but it has not been used much for research. Sycamore is a recent project and has several limitations as a general purpose simulator and there is clearly a lot of room for improvement.

Regardless of the tool used, more work on the experimental analysis of mobile robots algorithm would be extremely useful for many reasons. 1) As mentioned, complexity analysis is quite difficult to be done theoretically. Experiments would allow researchers to evaluate the efficiency of the algorithms under many scenarios, giving a way to experimentally compare different solutions, especially when a theoretical analysis is missing; 2) Local algorithms to achieve global tasks can be very complicated as often the local rules necessary to achieve a global goal could be counter-intuitive or simply difficult to find. Using a simulator to observe the resulting behavior of an algorithm could greatly help the design process; 3) Faults, imprecisions, unpredictable behaviors could be easily inserted into the system to observe their impact on known algorithms, and these observations could be the basis for a more advanced theoretical study.

1.1.3 Objectives of the Thesis

The main goal of this thesis is to improve the features of Sycamore and create new ones so that it could be more useful to the research community, in particular for experimentally observing previously unknown behaviors of swarms of robots when performing some tasks.

1.2 Contributions

We used Sycamore to execute mobile robot gathering algorithms and, in doing so, we developed new tools extending Sycamore, and we made some interesting observations on the algorithms themselves. Each algorithm we implemented is linked to specific new plugins and features for Sycamore: one algorithm especially requires the development of a batch simulator to be able to collect statistical results on success rates of the algorithms, another focuses mostly on being able to inject faults into the system and to observe their impact, the third requires the implementation of plugins that allow hybrid solutions with different robots executing different algorithms from hybrid initial condition settings. More precisely:

1. We extended Sycamore and developed a Sycamore Batch Simulator which uses Sycamore's Simulation Engine to repeat simulations in a batch. The Sycamore Batch Simulator uses a modest user interface to register the simulation setup and reuses the settings to perform every simulation in the batch, with the Sycamore Batch Simulator we overcame the problem and contributed to Sycamore development.

2. We designed and implemented a very simple probabilistic algorithm where a robot chooses the position of a random robot in its visibility range and moves to that position. We call this algorithm *Random Gathering*. The random gathering algorithm was used mainly to evaluate the Sycamore Batch Simulator, and we were not expecting good success rates. The results surprisingly exceeded our expectations. Interestingly, we identified the reasons for this success in the "Connected Initial Visibility Graph" plugin provided by Sycamore, that is, in the way Sycamore was choosing the initial, supposedly "random", configuration of the robots, which turned out not to be really random. To confirm our observation and to test the dependency of the success rate on the initial configuration, we created two new "Con-

nected Initial Visibility Graph" plugins to create different initial conditions. We then identified a clear link between the presence of clusters of robots in the initial configuration and the success of the algorithm.

3. We implemented a very well known algorithm [3] designed to converge to a point in semi-synchronous environments (i.e., in settings where all cycles are synchronized but only a subset of robots is activated in each cycle), and known to achieve exact gathering in fully synchronous settings. In the thesis, we call this algorithm *Exact Gathering*. Nobody had observed before the behavior of this algorithm in presence of faults. We considered crashes (i.e., robots that do not move for the entire execution of the algorithm) and we observed interesting behaviors both in the special 1 dimensional case of a line and in the general 2 dimensions setting of the plane. In few instances (essentially when the faults are close enough) all healthy robots still gather in a point in spite of the presence of faults. In many situations, as expected, we observe that the healthy robots eventually stop, forming a pattern inside the convex hull of the faulty ones. Finally, and unexpectedly, we also wittiness the robots reaching, as a limit behavior, oscillations between pairs of configurations. Note that our observations are limited to very specific situations (for example, faults placed in a particular shape, limited number of faults, etc.) and we did not exhaust all possible scenarios. As for the random algorithm, our main goal was to test Sycamore. But in doing so we have also unveiled some very interesting behaviors which should now be investigated further from a theoretical perspective.

   This study also led to the development of several new plugins and features, mainly to treat the injection of faults.

4. We implemented a combination of the previous two algorithms creating an hybrid execution where some robots follow the random algorithm, and some follow the

exact gathering algorithm. Neither group is aware of which, among the other robots, is following which algorithm. No faults are introduced in the system in this case. Again, we wanted to test Sycamore, but we also wanted to explore the possibility of two different algorithms working together towards a common goal. Note that random gathering is not in general very successful but requires extremely simple robots following an extremely simple rule (the robots just need to be able to perceive the positions of the others and move on a point). On the other hand, exact gathering is always successful but employs robots with slightly more capabilities (they need to make more complex computations). So, our idea was to see whether introducing very few, well placed, robots executing exact gathering would have a significant impact on the success rate compared to random gathering. We indeed observed that, even with as few as three well placed robots running Exact Gathering, the success rate of the execution can be substantially increased.

The hybrid algorithm made necessary the development of new plugins, mainly to allow placing robots in different formations in the same configuration.

1.3 Overview of the Thesis

We review some recent algorithmic results for mobile robots in Chapter 2, the chapter briefly introduces different visibilities, memories, schedulers and agreements. Chapter 4 focuses on the simulator "Sycamore" and its extension "Sycamore Batch Simulator". We briefly introduce the simulator's engine, plugins provided with Sycamore, the simulator's GUI and the plugins we created for Sycamore for the purpose of this thesis. In our experiments with Sycamore we observed some limitations and bugs which we have recorded for the use of future development and guide for users attempting to use Sycamore. We have introduced Sycamore Batch Simulator, we briefly explain its functions, GUI and provide a simple guide for users attempting to use Sycamore Batch Simulator. In Chapter 5 we

experiment with the *Exact Gathering* algorithm in presence of crashes. In Chapter 6 we experiment with the *Random Gathering* algorithm, first without crashes and then with a single crash. In Chapter 7 we evaluate the feasibility of executing more than one algorithm together to achieve a common goal, we experiment with the *Exact Gathering* and *Random Gathering* algorithms, and record our observations.

Chapter 2

Model and Background

2.1 The Robots

Mobile robots are autonomous entities capable of mobility. Mobility demands a space or universe in which the robots can move, the universe could either be a graph or a continuous region. In this thesis, we consider the robots are moving on a continuous universe $\mathcal{U} = \mathbb{R}^2$, represented as a Cartesian-coordinate system. Individually, mobile robots could observe their visible universe, perform computations and move. This *look-compute-move* sequence of states (which will be described below) is called the *robot's life cycle*. The robots execute this cycle infinitely, with a common goal; once the algorithm is completed the robots become inactive. The robots work together to perform some global task; and to do so they perform some form of communication, either explicit or implicitly. The robots may agree upon physical properties like the unit of distance, internal clocks, direction, Chirality and origin. To evaluate the strength of an algorithm the weakest assumption is usually made to identify the least requirements to accomplish a task, choosing the weakest assumption also promises better fault tolerance.

The robots could be solid (i.e., have a body) or be modeled by points. A solid robot could obstruct the view of another robot or cause collisions when two solid robots run

into each other. The robots that are a points do not pose such issues and the visibility is not limited by the position of robots, we have used robots that are points for our study in this thesis.

2.1.1 Life Cycle

Let $\mathcal{R} = \{r_1, \ldots r_n\}$ be the set of robots in the system. The total number of robots in the system is denoted by n.

A robot can be active or inactive. When inactive, a robot simply sleeps. When active, it performs the three phases corresponding to the three states described below:

Look State

A robot r_i observes the environment in the look state. The look state results in a snapshot, a snapshot contains the coordinates of other mobile robots observed in the visible universe of r_i. The snapshot is translated to the local coordinate system of r_i and not the global co-ordinate system, r_i is the origin of its own local co-ordinate system unless there exists an agreement on the origin.

Compute State

The robot computes a destination by applying the robot algorithm on the snapshot. The snapshot is the input for the compute state. The output of the compute state is a destination, the destination could either be a Cartesian-coordinate point or a null point, the null point signifies that the robot shall not move. The algorithm could also decide whether the final destination has been reached in which case the output is a null point and the robot enters into a indefinite sleep state.

Move State

The robot moves to the destination computed. The input for the move state is the destination point computed by the algorithm in the compute state, the destination could either be a Cartesian co-ordinate or a null co-ordinate. A null co-ordinate is ignored and the robot remains in the same position. A valid Cartesian co-ordinate initiates the robot's movement towards that co-ordinate in space, there is no inherent guarantee that the robot would reach that co-ordinate.

2.1.2 Visibility

The robot can observe its visible universe and generate a snapshot. The strongest assumption of visibility is *full visibility*, where r_i can observe the positions of any robot $r_j \in \mathcal{R}$.

When *limited visibility* is assumed, a common assumption is that of circular visibility: r_i can observe the position of all the robots in $\mathcal{U}$ which lie within a radius $V(r_i)$ from r_i (called *visibility radius*). Some other assumptions are seen in literature, like directional visibility and obstructed visibility. Circular visibility is used in this thesis and $V(r_j)\,\forall r_j \in \mathcal{R}$ is the same therefore the radius is simply denoted V.

The robot could have no multiplicity detection, weak multiplicity detection or strong multiplicity detection capabilities. Multiplicity detection is the ability to identify the density of robots on a point, if r_i could discern the number of mobile robots on the same point r_i is said to have *strong multiplicity detection*, if r_i could identify a point has more than one robot then r_i is said to have *weak multiplicity detection*, if r_i can not identify multiplicity on a point it has *no multiplicity detection.*

Given a system of robots with limited visibility, an important concept is the one of *visibility graph.* The nodes of the visibility graph are the robot co-ordinates in the spatial universe (note that the number of nodes do not necessarily have to be equal to the number of robots since more than one robot could occupy the same position in the universe).

The edges of the visibility graph connect the robots that are mutually visible to each other, it is assumed that the visibility radius of all the robots is V therefore all the edges are bidirectional. This assumption might not apply to directional visibility however it is not considered in this thesis. For most global tasks to be attainable, it is paramount that the initial visibility graph is connected.

2.1.3 Memory

The memory hosts snapshots and destination points of past cycles. Robots could have *unbounded memory*, in which case all the past snapshots and destination points are held; *finite memory*, in which case, the snapshots and destination points of a limited number of cycles in the past are held, or the robots could be *oblivious*, meaning that at the beginning of a cycle they do not remember anything about their previous cycles. In the thesis, we consider only the weakest case of oblivious robots.

2.1.4 Communication

The robots working together to perform a given task need to communicate. Communication could either be implicit or explicit. Implicit communication is the weakest assumption, the robots infer from the snapshot instead of an explicit method of communication, robots exercising implicit communication are said to be silent. Explicit communication is a stronger assumption and could be performed using different methods for example using lights; robots could have lights they could use the signal other robots or they could change the color of these lights to convey some information. In the thesis, we consider only implicit communication.

2.1.5 Agreements

The robots are autonomous entities, robots could have individual and exclusive understanding of physical properties like the unit of distance, direction, Chirality, origin, unit

of time and internal clock. Robots working together could agree on some of these properties before they begin their execution. The weakest assumption in agreements is no agreement, where the robots do not agree on any of the properties and the strongest assumption is complete agreement where the robots agree on all the properties.

2.1.6 Activation Scheduler

The activation scheduler is responsible for state switching. The activation scheduler escalates the robot from its current state to the next state. Some of the schedulers studied in literature briefly discussed in this section.

Asynchronous scheduler (ASync) is the weakest scheduler. The time spent on each state is unknown except that the time spent is finite, and this could vary for each robot in $\mathcal{R}$. Due to the lack of synchronicity between robots a robot moving to its destination in its move state could be seen by a robot in its look state. If the robots are *rigid* they are guaranteed to reach the computed destination at each cycle, otherwise they may be stopped during their movement at any time.

Fully Synchronous Scheduler (FSync) are at the other end of the spectrum. The robots are activated in rounds, each round corresponding to a cycle. All robots perform their actions simultaneously and atomically: they all look at the same time, they compute together, and they move in asycnhronized way. In Fully synchronous schedulers, they are always guaranteed to reach their destination.

Semi Synchronous Scheduler (SSync) is when the cycles are synchronized, as in fully synchronous schedulers, but not all robots are active in each cycle.

k-Bounded Scheduler limits the number of activations of other robots between two consecutive activations of r_i. A `2-bounded` scheduler will ensure that a robot $r_j \in \mathcal{R}$ will

not be activated more than 2 times between two successive activations of r_i.

Fair Scheduler guarantees that at time $t_i \exists t_j > t_i$ when r_i will be activated. The fair scheduler ensures that a robot is not indefinitely waiting to switch states. The `k-Bounded` scheduler is not necessarily fair, the `k-fair` scheduler is a fair variant of the `k-bounded` scheduler.

Slicing Scheduler is similar to the ATOM scheduler in that the activations are grouped into global round but only one robot is activated per group.

2.1.7 Faults

A faulty robot is a robot that does not behave as prescribed. Faults could be of very different nature and could influence any phase of the life cycle. Particularly interesting are crash and byzantine faults. A *crash* fault means that the robot becomes inactive and is never activated again. A *byzantine* fault means that the robot behave in some unpredictable way, for example, by moving to a different destination than prescribed. Faults can be transient or permanent, also they could occur before the beginning of the execution or during the execution..

Similar to faults are *inaccurate* computations or sensing capabilities. Inaccuracies could occur when perceiving the environment (returning a slightly incorrect snapshot) or when making a calculation (returning a slightly incorrect destination point).

2.1.8 Summary on Robots' assumptions

In this thesis, we consider oblivious robots in fully synchronous systems. The robots are equipped with limited visibility, they do not agree on a common coordinate system, nor on a unit distance nor on a common origin. The robots can detect multiplicities. Finally, the robots are silent and communication is achieved only implicitly by observing each

	Our robots
Scheduler	Synchronous
Memory	Oblivious
Visibility	Limited
Multiplicity	Detected
Communication	Implicit
Faults	Crash

Table 2.1.1: Summary on Robot's Assumptions

other's positions.

In the thesis we sometimes consider faulty robots. Whenever a robot is faulty we assume it is a crash fault. We also assume that the robot crashes permanently, before the execution of the algorithm.

Chapter 3

Related Work

In this Chapter, we review some algorithmic work on oblivious robots, with special emphasis on Gathering, the problem that will be treated in subsequent Chapters.

3.1 Pattern Formation

A pattern is represented as a set of positions or a geometric predicate. Pattern Formation is the coordination task of robots forming a predefined input pattern. Arbitrary Pattern Formation is the task of forming an arbitrary given pattern characterized by a set of positions. A particular pattern is the circle: Circle Formation is the task of positioning robots on the boundary of a circle and Uniform Circle Formation is the task of positioning them uniformly on the boundary of a circle. Arbitrary Pattern Formation is not solvable even with FSync and Chirality [32], Chirality is a form of common orientation where the robots agree on a "clockwise" direction. Arbitrary Pattern Formation is solvable in ASync with Consistent Compass [32], Consistent Compass is full agreement on orientation. Studies of Arbitrary Pattern Formation with OneAxis agreement are conducted in [32] where a solution is given to solve Arbitrary Pattern Formation in ASync with OneAxis when n is odd. Arbitrary Pattern Formation when n is even is also studied in [32] and it is proved that only patterns which do not have at least one axis of symmetry passing

through a vertex in the formed pattern is solvable when n is even with OneAxis in ASync. [32] extensively studies the limitations of Arbitrary Pattern Formation and similarities to the Leader Election Problem.

The Landmarks covering problem is similar to Arbitrary Pattern Formation, all the positions of the pattern are not only known to all the robots in their own coordinate system, but they are also visible throughout the computation. [34] presents a solution for Landmarks Covering with Chirality in ASync.

Circle Formation is trivially solved by moving the robots to the boundary of the smallest enclosing circle of all robots, even in ASync. Formation of an Uniform Circle in SSync is studied in [20, 21, 18], the later studies the problem with a guided trajectory. A solution to the Uniform Circle Formation problem in ASync is presented in [28] with Chirality and Guided Trajectory, the general problem of uniform circle formation has been recently solved and the solution is described in [30]. The problem of robots forming more than one pattern in a sequence is studied in [17].

3.2 Scattering and Covering

The problem of scattering is generally divided into two types, Removing Dense points and Covering a Region. The problem of Removing Dense Points is to achieve a plain configuration from an arbitrarily random initial configuration with possible dense points, a plain configuration is one where there exists no dense points, a dense point is a position in the universe occupied by more than one robot at the same time. A simpler problem is the Split problem wherein robots which are at mutually different positions at time t will remain at mutually different positions at time $t' > t$. The Split problem is studied in [22, 39]. Split is not solvable even in FSync with deterministic algorithms. [22] presents a probabilistic solution to solve Split in SSync with Unlimited Visibility and multiplicity detection. A solution for Split in SSync with Limited Visibility and multiplicity detection

is presented in [39]. Scattering problems associated to Covering a Region are studied in [13, 28, 5]. Studies in [13] consider a line, they have presented a solution in which robots converge toward a uniform covering of the line in SSync. Attempts to solve Covering a Region over a ring is carried out in [28], the study proves the impossibility of the solution without Chirality in SSync even with unbounded persistent memory and Unlimited Visibility radius. The solution for uniform covering of the ring with chirality presented in [28] converges toward the uniform covering of the ring, the solution terminates with successful completion of the uniform covering when the robots posses knowledge of the final inter robot distance on the ring. For a survey on scattering in linear environments under limited visibility settings, see also [27]. Most research on coordination task assume the robots are within the region where the task is to be performed, [5] studies the problem of "Filling of Orthogonal spaces" wherein robots enter the region they are to fill from a point called "door".

3.3 Gathering

Gathering is a simple coordination task where in all the participating mobile robots should gather at the same position in the universe from an initial configuration where they were placed at arbitrarily random positions. Gathering of robots with a FSync scheduler and Unlimited Visibility is easily solved with the Center of Gravity target function discussed in [11], similarly, Gathering of robots with an ASync scheduler and Limited Visibility is trivially solved with a Consistent Compass [31]. A primary research focus has been to identify the least required agreement on local coordinate systems and multiplicity detection to achieve Gathering in different schedulers. We know that Gathering is deterministically unsolvable in SSync, therefore also in ASync, in the absence of multiplicity detection and any agreement on local coordinate systems [47].

Gathering with simpler agreements on the coordinate system in SSync and Unlimited

Visibility, using Chirality and Tilted Compass, is studied in [38]. Gathering with no agreement on coordinate system but with a strong multiplicity detection and Unlimited Visibility is studied in [23], this solves Gathering only when n is odd.

Gathering is ASync is understandably more difficult, studies with Weber Point in [53, 4, 24] prove that Gathering is possible when a point is chosen which is invariant to mobile robot movements in Unlimited Visibility, the flip side is that computing the Weber Point when $n \geq 5$ is not possible. Gathering with only multiplicity detection is studied in [9], this solves Gathering only when $n \geq 5$. Feasibility of combining Weber Points for $n < 5$ and multiplicity detection for $n \geq 5$ in ASync with Unlimited Visibility is studied in [10]

There have not been many studies on Gathering with Limited Visibility in literature. In [31], a simple solution for Gathering with Consistent Compass is presented. Other studies dwell on the possibility of achieving Gathering with lesser agreements on local coordinate system like [49] which presents a solution to achieve Gathering in SSync with Chirality and eventually Consistent Compass in Limited Visibility. Studies to extend this solution to ASync are conducted in [48].

Rendezvous is a special case of Gathering with only $n = 2$ robots. Extensive study on Rendezvous is undertaken in [40]. Rendezvous, similar to Gathering, is trivially solved in FSync and a slight modification to the Center of Gravity target function known as the Move-To-Half target function studied in [11]. Note that any solution for Gathering will invariably solve Rendezvous. In depth studies on Rendezvous with Tilted Compass is carried out in [40], Rendezvous is solvable with Static Tilted Compass in SSync and ASync, with Variable Tilted Compass in SSync and with a Semi-Variable Tilted Compass in ASync. It is currently unknown whether Rendezvous is solvable in ASync with a Fully Variable Tilted Compass. It is known from [50] that Rendezvous is impossible in SSync (therefore also in ASync) without an agreement on local coordinate systems even with strong multiplicity detection.

Convergence is another coordination task closely related to Gathering in which the robots do not move exactly to the same position in the end but instead get closer and closer to some common point without ever reaching it. Some Gathering solutions like the Center of Gravity target function presented in [11] perform Convergence with no modifications to the algorithm in SSync and ASync and Unlimited Visibility. Convergence is observed in the Rendezvous problem in SSync [50] and ASync [11]. The Point Convergence algorithm introduced in [3] solves Convergence in Limited Visibility and no other agreements or multiplicity is necessary. The convergence time with Point Convergence is studied in [19] and some variants of the algorithm is presented in [44]. Similarly to Gathering, only a few studies have been conducted on Convergence in Limited Visibility: Convergence in a Non-Convex region with Limited Visibility and FSync scheduler is studied in [36], Convergence with a partially ASync Scheduler and Limited Visibility is studied in [45] and Convergence with a 1-Fair ASync Scheduler and Limited Visibility is studied in [41]. Studies like that in [14] undertake the cause of decreasing the convergence time. For a survey on Gathering and Convergence by robots with limited visibility, see also [26].

Near-Gathering is closely related to Convergence, the robots do not Converge close to a common point but instead, the robots move to a formation wherein they are ϵ distance away from each other. Near Gathering could be simplified as a Convergence where no robots occupy the same position at the same time in the entire execution. Near Gathering is achievable with slight modifications to the Center of Gravity target function and Unlimited Visibility in an ASync Scheduler setting [11]. Near Gathering is only achievable in SSync and ASync with the knowledge of the number of robots in the configuration. A slight modification of the Point Convergence algorithm is studied to solve Near Gathering in SSync and Limited Visibility with no other agreements on local coordinate systems [3]. Near Gathering in ASync can be achieved with Consistent Compass and Limited Visibility in the Infinity Norm [46].

In most of literature, studies focus on Gathering with an assumption that there could occur no faults or errors. It is however necessary to consider situations where the measurements observed by the robot in the Look phase is flawed or the robot itself is misbehaving. A detailed study on robot Gathering and Convergence with inaccurate measurements is conducted in [12], we see that Gathering in FSync of $n = 2$ robots is impossible in FSync even with Consistent Compass and multiplicity detection with inaccurate measurements and Gathering with $n > 2$ robots is impossible even with Consistent Compass and multiplicity detection in SSync. Studies on Convergence in [12] show that it is possible to achieve mobile robot Convergence with a modified version of the Center of Gravity target function called the Restricted Center of Gravity in SSync with Unlimited Visibility. In the Undimensional space, Convergence is even possible in ASync with Unlimited Visibility using Restricted Center of Gravity target function. Convergence with radial error in measurements is studied in [43] with a FSync Scheduler and Limited Visibility.

Misbehaving robots or Faulty robots are classified into two categories: Crashed robots, which permanently stop executing their life cycle before the termination of the algorithm, and Byzantine robots, which perform more complex actions. Gathering in ASync scheduler with Crashed robots, Consistent Compass and Unlimited Visibility is solved very simply by moving to the robot which is north most and east most, a variation of the Gathering algorithm presented in [31]. The solution for Gathering presented in [10] successfully performs Rendezvous in the presence of a Crash with only Multiplicity Detection and Chirality. Gathering with one Crashed robot in a configuration with $n \geq 3$ in SSync Scheduler is achieved in [2]. A powerful solution for Gathering with strong multiplicity detection and Chirality in SSync scheduler when $f < n$ is discussed in [6] (f is the number of Crashed Robots). Studies on Byzantine robots are limited, these are a few research we see in literature [2, 7, 8, 37].

Note that all the existing work dealing with faulty robots considers the full visibility

setting.

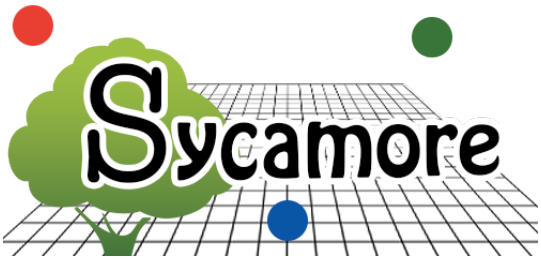
3.4 Other Models

Several variant of the main model have been recently introduced. One such variant is given by the availability of lights to the robots. The model of luminous robots has been introduced to study the impact that little memory can have on the robots' computability power. In this model the robots are equipped with a light that can take a constant number of different colors. The light is typically visible to the robots that are observing and stays on from a computation cycle to the next (i.e., it is persistent in spite of robots' obliviousness). The availability of lights has been studied to see the impact on the various synchronicity models [16] as well as in the study of specific problems (e.g., see [42, 33, 51]), for a recent survey, see [25]. Another variant of the model studied in the literature consists of solid robots. A solid robot is considered to have a body and its main consequence is that robots can obstruct each other's visibility. Solid robots have been introduced in [15] and subsequently studied, especially in connection with the gathering problem [1, 35].

Chapter 4

Sycamore

In this Chapter, we introduce Sycamore [52] and Sycamore Batch Simulator, we describe the new plugins developed in the thesis and we indicate possible future developments. This chapter introduces the simulator, briefly explains its functions and describes its extention, the Sycamore Batch Simulator. The simulation environment is explained in Section 4.1, Sycamore plugins in Section 4.2, Sycamore's GUI in Section 4.3, Plugins we have created for Sycamore in Section 4.4, Sycamore Batch Simulator in Section 4.5 and it concludes with our observations and suggestions to the developer and whosoever desires to work with Sycamore in Section 4.6.



4.1 Sycamore Simulator

A simulator is used to setup and test virtual entities representing real world objects and observe their behavior. Properties of mobile robots customizable in Sycamore are: algorithm, visibility, memory, agreement and scheduler. The simulator is implemented in Java, the simulator can be used in any operating system which runs the java virtual machine (JVM). Sycamore provides a live preview of the simulation, animated objects are controlled by Java Monkey Engine (JME). JME is a game engine that runs on Java Virtual Machine, it helps create 2D and 3D graphics and provides tools to control the objects. Sycamore allows customizing robot properties by programmable modules of code called "plugins", Sycamore uses Java Simple Plugin Framework (JSPF) to interact with plugins. Sycamore is an open-source application, the code is available for developers to learn and expand.

The engine is a core component which users cannot modify with "plugins", it presides over and ensures proper interaction of all components in Sycamore. The engine is responsible to keep components updated with user preferences and choices, it is responsible to operate the scheduler thread which activates different phases in the robot's life cycle, it provides snapshots to each robot during its look phase, it moves the robot during its move phase and updates any measure listening to the simulation.

4.2 Plugins

Sycamore Plugins are programmable modules of code which are added to the simulator, a plugin is a customization of a robot property. The Sycamore Plugin SDK allows to implement customization, the newly created plugin is packed into a JAR file to use with Sycamore. Sycamore provides basic implementation of every property to help avoid rework, these abstract methods can be extended. The basic implementation handles simple interactions with the GUI to provide display information for the plugin, e.g.

plugin name and plugin type, it also provides plugin specific functions, e.g. setting fixed measure units for agreements.

Sycamore allows specific plugin configuration through the plugin configuration window (Figure 4.2.1). The plugin configuration window can be accessed by double clicking the imported plugin in the plugin list panel. Specific plugin configurations are useful, they allow simple customizations using GUI objects like buttons, text field and sliders. Plugin categories and plugins provided by Sycamore are discussed below, plugins provided by Sycamore are readily available for download and use:

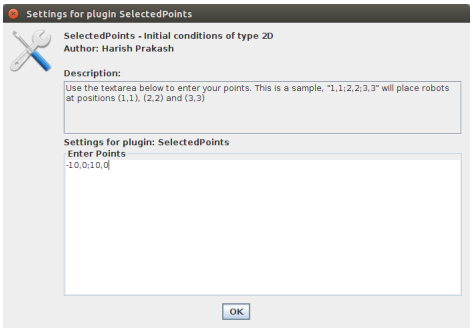


Figure 4.2.1: Plugin Configuration Window

Visibility

The default visibility is "Complete Visibility", robots see each other in spite of the distance between them. Sycamore provides the following plugins for immediate use in 2D, *Circular visibility*, *Directional visibility* and *Squared visibility*. The plugin configuration window for every visibility plugin allows the user to choose the visibility radius. The simulation preview allows the users to visualize the visibility boundary of each robot in its respective shape (circle, cone or square) (Figure 4.3.5). Sycamore Plugin SDK shares user defined visibility radius across all visibility plugins.

Memory

The default robot memory is "Oblivious", robots will not remember snapshots or destinations from past cycles. Sycamore provides the following plugins for immediate use: *Bounded Memory* and *Largest Memory.* Bounded Memory allows every robot to remember snapshots and destinations from limited cycles in the past, Largest memory allows robots to remember snapshots and destinations from every cycle in the past.

Scheduler

There is no default scheduler, Sycamore does not assume a default scheduler unlike other properties. Sycamore provides the following plugins for immediate use in 2D, *Fully Synchronous Scheduler*, *Semi Synchronous Scheduler* and *Asynchronous Scheduler*. The fully synchronous scheduler activates every robot at each scheduler thread, the move phase in fully synchronous scheduler will not be interrupted, any robot which has completed the move phase will remain in the sleep phase till all robots have completed their move. The semi synchronous scheduler activates robots in groups, the inactive groups remain in the sleep phase till active groups complete their cycle. The asynchronous scheduler is sporadic, robots may wake up, compute and move at any time, fairness can be configured in the plugin configuration window (Figure 4.2.1).

Measures

A measure is not a robot property like other plugins, it monitors every robot over the period of the simulation and generates a report called the measure. Measures are plugins which can be created with the Sycamore Plugin SDK. Sycamore does not generate a default measure. Sycamore provides the Elapsed Time measure and the Average Distance measure. The Elapsed Time measure prints the time taken to complete the last simulation. The Average Distance measure prints the average distance travelled by all the robots in the last simulation.

4.3 Graphical User Interface (GUI)

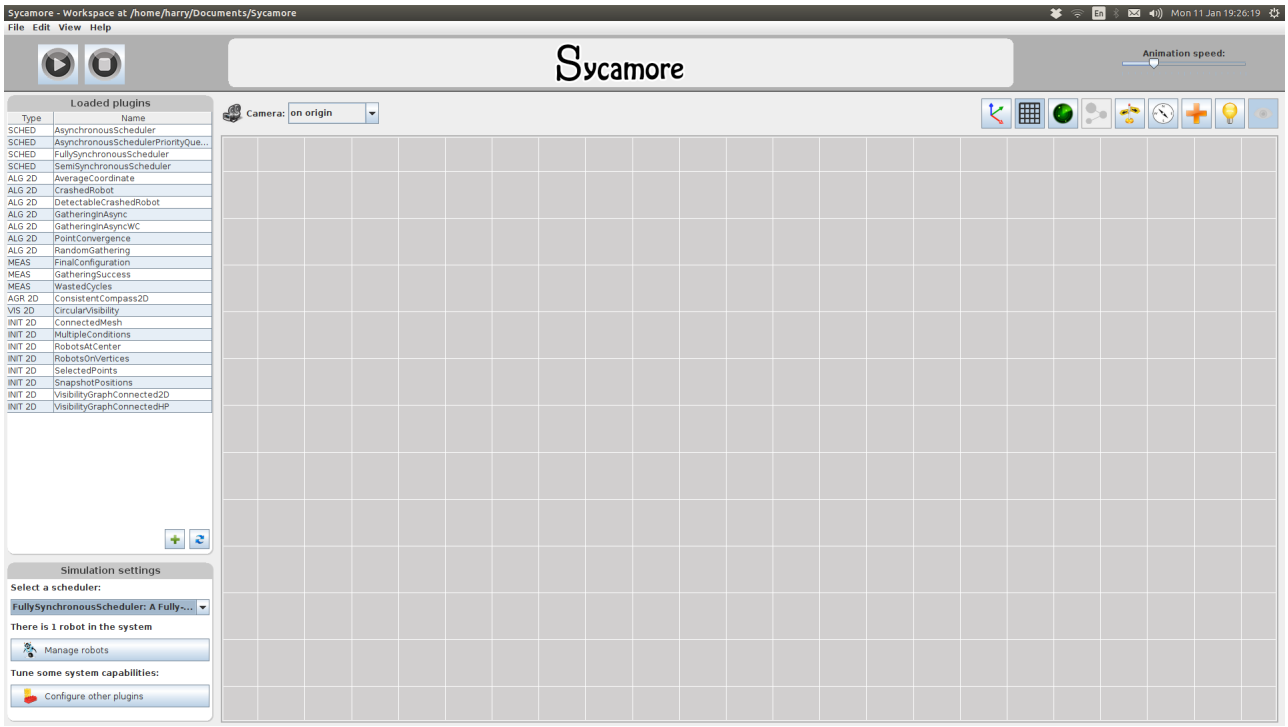


Figure 4.3.1: Sycamore Main Window

The user interacts with Sycamore using the GUI. The GUI allows the user to add plugins, configure plugins, refresh plugins, configure mobile robots, setup simulation, modify simulation preferences, control the simulation state, preview the simulation, control animated objects in the preview and save/load simulation.

Common Plugin GUI

Plugins can be added, configured or refreshed. The plugin list panel is immediately accessible in the simulator main window (Figure 4.3.1). The plugin list panel is initially an empty list which can be populated by using the "Add Plugins" button. The list can be refreshed using the "Refresh Plugins" button. The list displays the plugin name and plugin type for each plugin, plugin type refers to the robot property provided by the

plugin. When the plugin is listed, it is ready for simulation. Plugins can be specifically configured using the plugin configuration window (Figure 4.2.1), it is accessed by double clicking the appropriate plugin.

Configure Mobile Robots

Mobile robot configuration is accessed through the "Manage Robots" button in the simulator's main window. The robots are grouped into robot lists for ease of use, each list can be assigned an algorithm, color, speed, lights and the number of robots. A list should contain at least one robot and may contain up to 100 robots. Every algorithm plugin added to Sycamore will be listed for selection besides each list, the algorithm associated to the list will be executed. The robot color does not affect the robots, they are used to distinguish the lists in simulation preview. The robots in their move phase travel at the preset speed. The lights may be used by algorithms to explicitly communicate between robots. The configuration to the list is mutually exclusive, different lists may have different algorithm, color, speed, lights and number of robots but every robot in the list is configured the same. Visibility is common to all the lists.

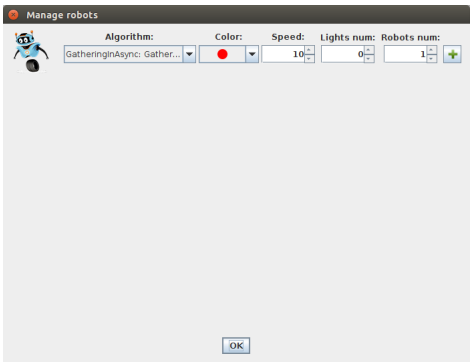
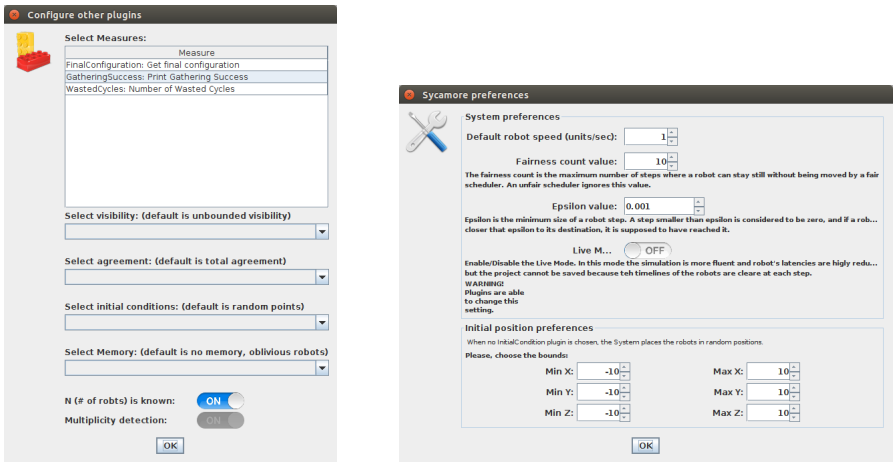


Figure 4.3.2: Robot Configuration Window

Common Simulation GUI

The minimum setup for a simulation in Sycamore is selecting a scheduler and setting up at least one robot list. Any configuration made out side the robot configuration window is applied to all the robot lists. Robot properties like visibility, initial condition, memory, agreement and simulation measures are configured in the "Other Plugins Configuration Window" accessed through the "Configure Other Plugins" button in the Sycamore Main Window. The scheduler is configured using the scheduler selection combo box in the Sycamore Main Window.

Simulation parameters like *Default Robot Speed*, *Fairness Count Value*, *Epsilon Value*, *Live Mode* and *View Port Size* are configured through Sycamore Preferences Window accessed through the Edit Menu. The default robot speed is a value in units/sec, it is an upper limit on the speed at which robots may travel. The fairness count value is accessible through scheduler basic implementation in the Sycamore Plugin SDK, this value is used to setup fairness count of asynchronous schedulers. Epsilon value is used to fix floating point imprecisions commonly seen in digital computers, it is the smallest distance less than which robots will not move. Epsilon value may be used in algorithms to provide simple error correction. Live Mode is a faster mode of simulation, it is turned off by default. Sycamore performs additional computations at each scheduler step to store timelines, the timelines can be saved anytime in between the simulation, this overhead reduces the performance of the simulator. Turing on Live Mode deactivates continuous timeline monitoring. Timeline is a data structure created to save Sycamore simulations, it saves every position of every robot from time t_0 till t_{end}. View Port is a configurable territory in Sycamore, the shape of View Port is a square in 2D and a cube in 3D. The center of View Port is the center of the universe which is position (0, 0). Sycamore Plugin SDK shares the View Port boundary to plugin implementations, View Port can be used to perform bounded computations in the universe.



(a) Other Plugins Configuration Window

(b) Sycamore Preference

Figure 4.3.3: Simulation GUI

Visualization

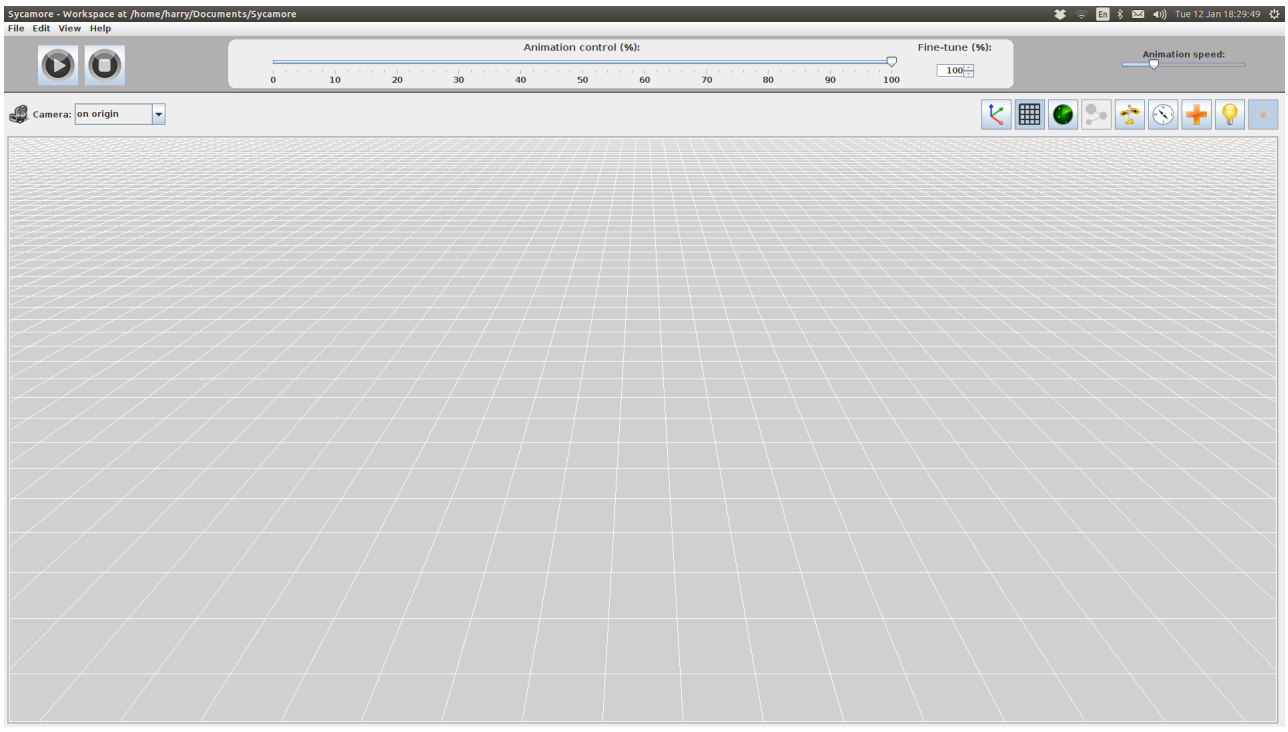
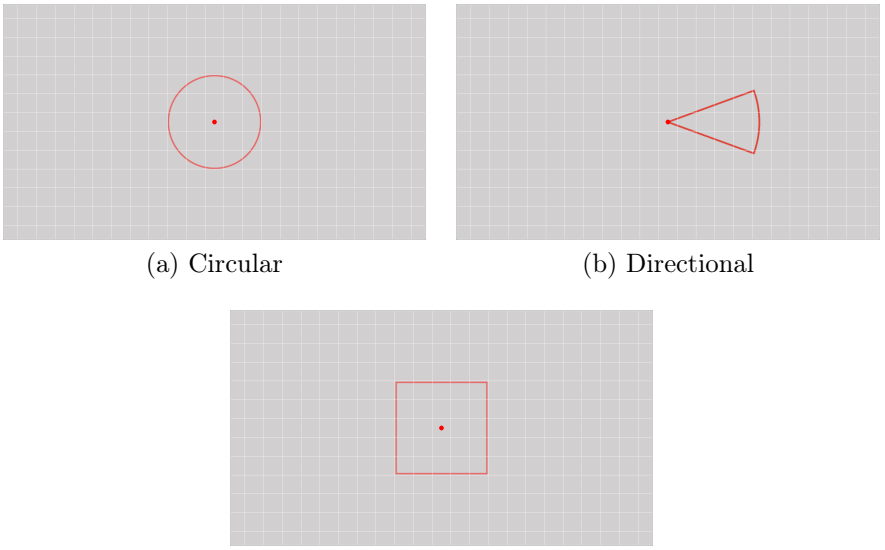


Figure 4.3.4: Sycamore Visualizer

Once the simulation begins it may be paused, stopped or completed; these are the states of the simulator. The Sycamore Main Window provides easy controls to play/pause or stop the simulation, the controls will become active once a scheduler is selected. In execution, that is when the simulation is playing, the user can preview the simulation in the preview window. Robots are represented as discs in 2D, their positions in the preview reflect their current position in the simulation. The visibility boundaries are delimited with a geometrical shape representing the kind of visibility, circle for circular visibility, cone for directional visibility or square for squared visibility (Figure 4.3.5). Lights, their colors and switching are graphically shown in the preview. The animation controls accompany the preview window, these controls do not affect the simulation but help users better understand the simulation. Most animation controls are toggles that show global coordinate axes, grid, visibility range, robot direction, local coordinate system and lights. A slider is provided to increase/decrease the simulation speed while in preview.

Visualizer is a tool provided with Sycamore, it is a stand alone application. Visualizer can load simulations (timelines) saved in Sycamore, it is helpful when the user wants to review a simulation or robot behaviors. Visualizer decodes the timeline and replays the simulation, users can control the speed of playback in Visualizer.



(a) Circular

(b) Directional

(c) Squared

Figure 4.3.5: Visibilities

4.4 Extended Plugins

Sycamore is programmable, it allows us to setup simulations to suit our need. We have implemented the following plugins for the purpose of our experiments,

1. **Crashed Robot**

   Crashed robot is a simple algorithm, on the first execution of the algorithm the robot terminates and enters a permanent sleep mode. The crashed robot does not move, it remains in the position it was at t_0.

2. **Point Convergence - Algorithm**

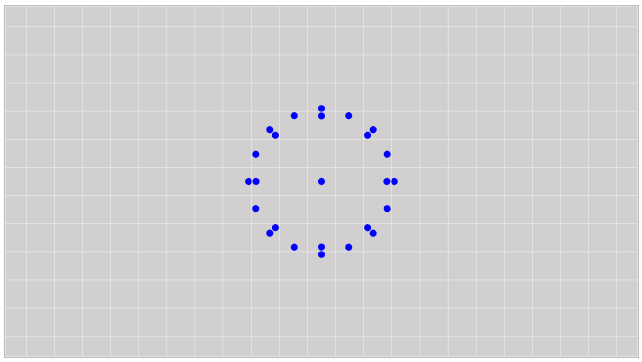


Figure 4.4.1: Robots executing Point Convergence

Distributed memoryless point convergence algorithm for mobile robots with limited visibility is a convergence algorithm we have used in our experiments. The algorithm makes all the robots gather to a single position in Fully Synchronous scheduler, the robots converge in a Semi Synchronous scheduler.

3. **Random Gathering - Algorithm**

   This is a simple algorithm we invented to test different initial conditions and their effect on mobile robot algorithms. The mobile robots pick a random robot from their snapshot and move to its position.

4. **Center Of Graph - Initial Condition**

   The plugin identifies smallest enclosing circle of all robots currently in the Universe and places new robots at the center of the circle. This does not guarantee connected visibility graph.

5. **Connected Mesh - Initial Condition**

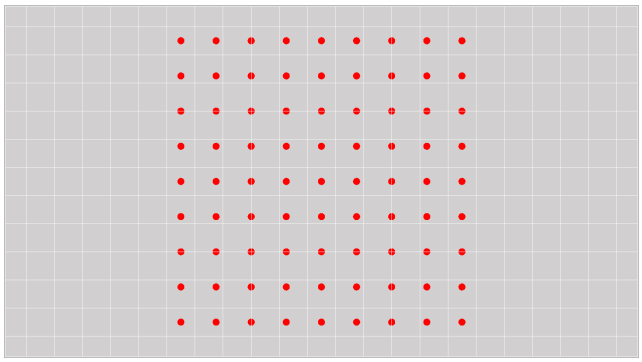
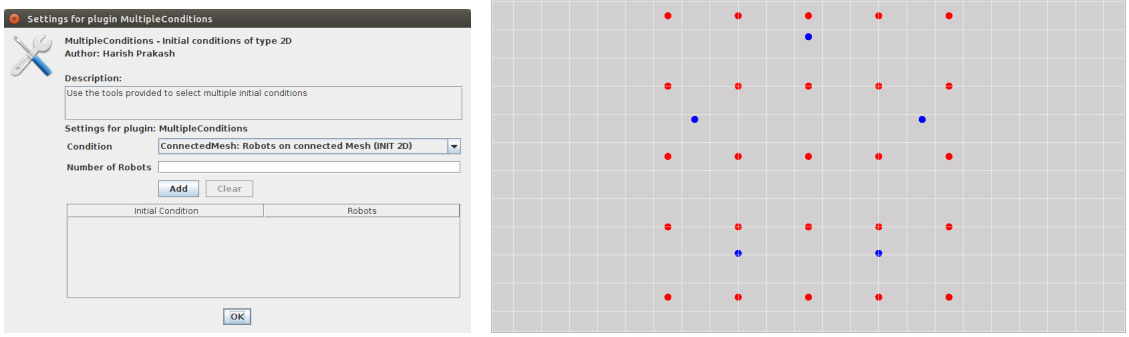


Figure 4.4.2: Robots placed on a Mesh-Like Initial Connected Visibility Graph

Robots are placed to create a skeleton and then fill the View Port, the skeleton is a mesh like formation where each node of is occupied by a robot. The distance between any two rows in the skeleton is V and the distance between any two columns in the skeleton is V. The purpose of the skeleton is to cover the View Port such that there does not exist a position in the View Port that is not seen by a robot in the skeleton. Once the skeleton is complete, robots are randomly placed in the View Port. This guarantees a connected visibility graph.

6. **Multiple Conditions - Initial Condition**



(a) Multiple Conditions Plugin Configuration Window

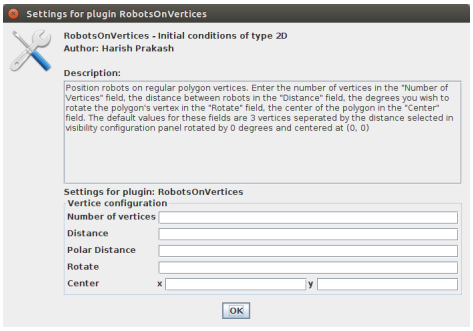
(b) Robots Placed in Mesh-Like Initial Connected Visibility Graph and Vertices of a Polygon

Figure 4.4.3: Multiple Conditions Plugin

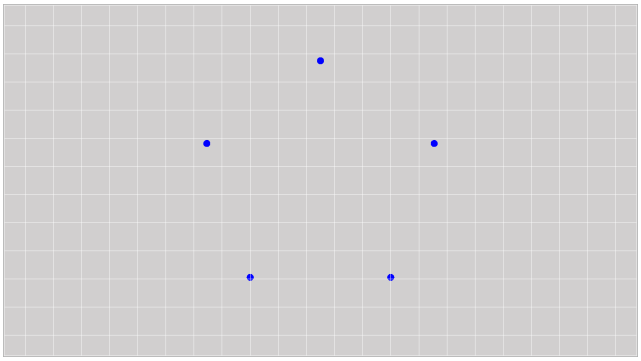
This initial condition combines more than one initial condition, this allows switching initial conditions when the robots are being placed. The switch is triggered by

the number of robots. The user should use the plugin configuration window, accessed by double clicking the "Multiple Conditions" plugin in the plugin list panel. In the plugin configuration window, the user should choose an initial conditions and declare the number of robots the user wishes to place using the selected initial condition; then repeat the process for multiple initial conditions.

7. **Robots on Vertices of a Polygon - Initial Condition**



(a) Robots On Vertices of a Polygon Configuration Window



(b) Robots on Vertices of a Pentagon

Figure 4.4.4: Robots On Vertices of a Polygon

Robots are placed on vertices of a regular polygon. A polygon is a closed figure with at least three vertices, the distance between every vertex is the same in a regular polygon. The default setting for the plugin without any customization is, the robots on three vertices and the distance between them is $> 2V$. The number of vertices, distance between the vertices, polar distance from the center to the vertex and the center of the polygon can be customized using the plugin configuration window.

8. **Selected Positions - Initial Condition**

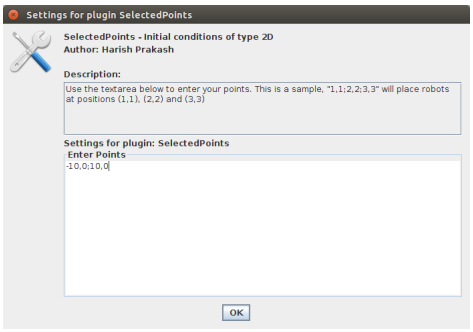
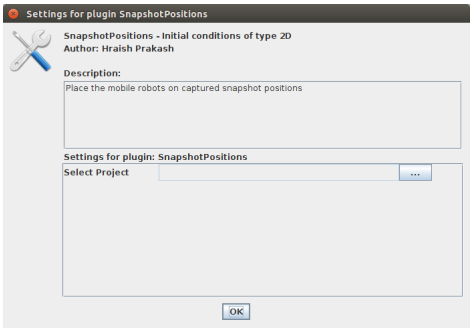


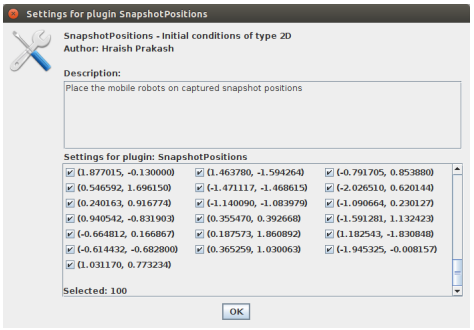
Figure 4.4.5: Selected Positions Plugin Configuration Window

Robots are placed at user defined positions, the user must declare the positions in the plugin configuration window. The default position if the user does not specify other positions is (0, 0). To specify a position whose x-coordinate is 5 and y-coordinate is 3 the user should type $(5, 3)$; in the plugin configuration window, more positions can be appended after the semicolon or in a new line.

9. **Snapshot Positions - Initial Condition**



(a) Timeline File Selection



(b) Position Selection

Figure 4.4.6: Snapshot Positions Plugin Configuration Window

Robots are placed at positions in a timeline file. The plugin configuration window allows choosing the timeline file, the selected file is decoded and the user is allowed to choose specific positions using check boxes.

10. **Visibility Graph Connected - Initial Condition**

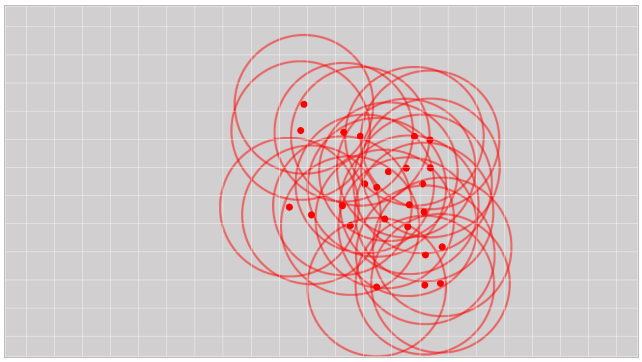


Figure 4.4.7: Visibility Graph Connected

This is an edited version of the original Visibility Graph Connected plugin, changes are made to guarantee all robots lie within the View Port.

11. **Final Configuration - Measure**

    Prints a list of every occupied position in the console at the end of the simulation. If more than one robot occupy the same position, the position is printed twice. The list is automatically printed at the end of the simulation, the plugin does not work if the "Stop Simulation" button is used to terminate the simulation.

12. **Gathering Success - Measure**

    Prints a flag identifying the success of gathering to the console at the end of the simulation. The number 1 is printed to denote a successful gathering and the number 0 is printed to denote a failed gathering. After multiple simulations these numbers can be added to identify number of successful gathering and the number of entries is the number of simulations performed.

4.5 Sycamore Batch

Sycamore Batch Simulator is an extension to Sycamore, it merely interfaces user preferences to the original simulator. Sycamore Batch Simulator is designed to perform

simulations in batch, a batch is a set of simulations with the same settings and robot lists.

4.5.1 Motivation

Sycamore is capable of performing simulations with mobile robots, it allows customizing their properties and helps record useful measures from the simulation. Sycamore is designed to perform one simulation at a time, if the user wishes to execute more than one simulation the user must reconfigure the simulator each time. The experiments we perform are statistical in nature, we need to perform thousands of simulations. Sycamore is focused on visualizing the simulation but we require a simulator to automate multiple simulations, the short comings of Sycamore or the advantages of the batch simulator are;

1. **Perform Multiple Simulations**

   Sycamore Batch Simulator allows users to setup the simulation once per batch, the simulator applies the same settings for each simulation in that batch. The duplicated setup does not necessarily provide the same result in every simulation, if the robots are randomly placed or a scheduler other than the fully synchronous scheduler is used then each simulation will be unique.

2. **Simulations are Faster**

   The batch simulations are quicker than simulations in sycamore, this is possible because of these three features. One, the user setup is not repeated for each simulation. Two, the JME (Java Monkey Engine) is not fully engaged since no visualization is required. Three, the simulations in batch run on live mode which disables continuous timeline monitoring. Each feature boosts the simulation speed greatly.

3. **Automation**

The purpose of a batch simulator is automation, it allows users to setup simulations and configure the batch. Sycamore Batch Simulator allows automation of multiple simulations, the simulator is also extensible and allows developers to add more components which may allow further automation, e.g. collect timelines to preview in the visualizer, save simulation and batch settings to a file.

4.5.2 Simulator

Sycamore Batch Simulator is efficient and easy to maintain or improve, Sycamore Batch Core and Sycamore Batch GUI are two individual applications that together make Sycamore Batch Simulator.

Sycamore Batch Core (Core) interacts with Sycamore and Sycamore Batch GUI (BGUI) interacts with the user, the applications interact with each other using event driven communication to increase performance and responsiveness. Similar to Sycamore, Sycamore Batch Core and Sycamore Batch GUI is developed in Java and Java Swing respectively. Sycamore Batch GUI provides usable interface components like text boxes, buttons, combo boxes and sliders. Sycamore Batch GUI is wholly operated by the user, it does not run any background tasks or fire events when the user is in the process of setting up the simulator and the batch. When the user is ready to start the simulation, Sycamore Batch GUI locks the user interface to prevent further changes, encapsulates user settings and registers with Sycamore Batch Core. The registration is simply the GUI sending user settings and registering as a listener to Sycamore Batch Core events. GUI responds to event notifications from Core by displaying appropriate status messages in the Console.

Sycamore Batch Core is responsible for receiving user settings, apply work-arounds, setup sycamore, initiate sycamore simulation, listen to sycamore states, announce results to listeners and repeat the simulations. Once Core receives user settings from the GUI, it begins the batch by setting up a counter representing the number of simulations in the

batch. Core performs setup for each simulation in the batch, the setup is tweaked with these work-arounds to overcome limitations in Sycamore or enhance the simulations;

1. **Apply Visibility**

   Applying visibility plugins on all robots requires multiple steps in the original Sycamore GUI, the steps are discussed in Chapter . The purpose of the batch simulator is to avoid user interaction in each simulation. To seamlessly apply visibility, the batch creates each robot in each simulation individually and applies the visibility plugin. The method is not optimized but achieves the expected result without modifying Sycamore.

2. **Animation Speed**

   Simulations in Sycamore are impacted by GUI preferences like the animation speed, a robot's speed is proportional to the animation speed slider value in Sycamore. The batch simulator resets the slider to max at each simulation to achieve best results.

3. **Turn on Live Mode**

   The live mode is useful for batch simulations when the purpose of the batch is to gather measurements and not timelines, timeline monitoring demands much resources which is freed in Live Mode speeding the simulations.

4. **Apply cool down time**

   Sycamore runs multiple threads in parallel, as a result Sycamore Batch Core cannot immediately begin the next simulation in a batch after one ends. A cool down time of 5 seconds is used pad simulations. Repeated experiments showed us at least 1 second between simulations is reasonable, we have selected 5 seconds which includes considerable buffer to avoid lags in different machines. Using a cool down timer helps avoiding complex thread monitoring. The ideal solution would be to modify Sycamore to ensure thread termination before simulation end notification.

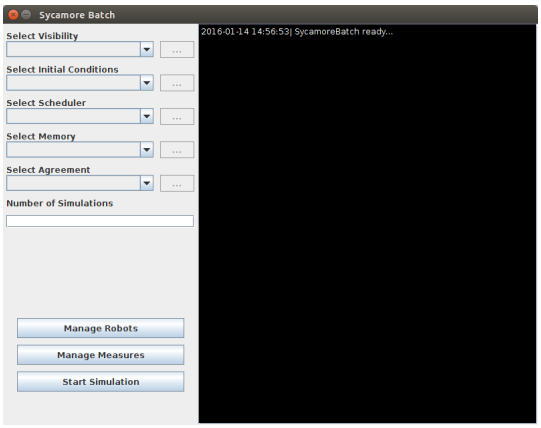


Figure 4.5.1: Sycamore Batch Simulator

After a simulation is setup, Core registers with Sycamore to be notified when the simulation ends and initiates the simulation. At the end of each simulation Core notifies all the listeners, applies a cool down time to prevent overlapping threads from causing inconsistencies, the batch counter is reduced and the process repeats till the counter reaches zero. At the end of the entire batch, Core notifies all the listeners.

Isolating Sycamore Batch Core and Sycamore Batch GUI allows independent maintenance and development. Sycamore Batch Core is designed to be consumed as an API which allows integration with other applications. The API design of Core allows choices, for example, it is possible to create more than one GUI for the batch simulator. An extension library is provided with Sycamore Batch Core, it provides methods which create simple features. The only method available in the extension library is “toggle live mode on/off”, it can be updated to include more methods in the future.

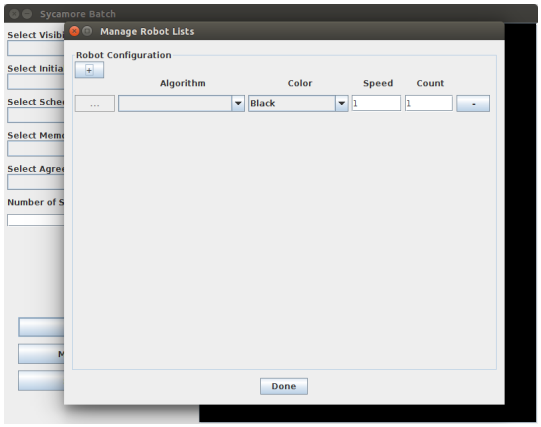
4.5.3 Sycamore Batch GUI

Sycamore Batch GUI provides useful components like text boxes, buttons and combo boxes to setup simulations. The user interface is simple, all the plugins are selected by combo boxes similar to Sycamore. Every plugin setup, except algorithms and measures,

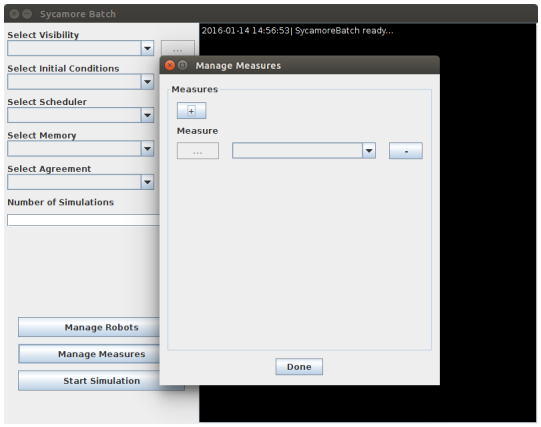
are available from the main window (Figure 4.5.1), algorithms and measures have individual configuration window to allow multiple selection. The console is a preview terminal which displays status messages from Sycamore Batch Core, the console is read-only. Minimal requirements to begin a batch is: selection of a scheduler, setup of at least one robot and entering the number of simulations in the batch. The Robot Management Window (Figure 4.5.2a) is accessed by clicking the "Manage Robots" button in the main window, the user may configure up to five robot lists in this window. Robot list selection is similar to Sycamore, lights are not allowed in the batch simulator due to limitations in Sycamore. The Measure Management Window (Figure 4.5.2b) is accessed by clicking the "Manage Measures" button in the main window, the user may configure up to five measures in this window. The limited number of robot list and measure selection is setup to simplify the GUI.

Plugins can be configured (Figure 4.5.2c) similar to Sycamore, the GUI provides a "settings" button besides every plugin selection combo box. The "settings" button is initially inactive, the button is activated when the user selects a plugin which allows plugin specific configuration. The "settings" button is also available in Robot Management Window and Measure Management Window.

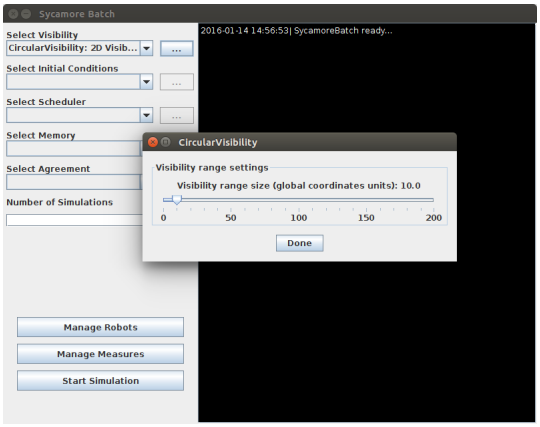
Sycamore Batch GUI currently does not support adding plugins, plugins can only be added using Sycamore.



(a) Robot Management Window



(b) Measure Management Window



(c) Plugin Configuration Window

Figure 4.5.2: Sycamore Batch Simulator

4.6 Future Work in Sycamore

Working with Sycamore and developing Sycamore Batch Simulator has allowed us to test the application and identify some areas of development, we have compiled a list of suggestions to guide future developers and inform users of what they could expect.

1. **View Port and Visibility Graph Connected**

   The "Visibility Graph Connected" plugin provided with Sycamore places robots inside View Port when robots have complete or full visibility. If a visibility is

selected, the plugin ignores View Port boundaries and places the robots even outside the View Port.

2. **Apply Visibility to Robots**

   The application of visibility to robots is faulty in Sycamore, one would expect to select a visibility and place the robots and hope the visibility would be applied to all the robots but this does not happen. To apply visibility to robots the following steps should be followed in the same order; first configure a single robot through the "Robot Configuration Window", second use the "Other Plugins Configuration Window" to setup the visibility, finally return to the "Robot Configuration Window" to setup other robots.

3. **Visibility Circle in the Preview Window**

   The red circle, cone or square denoting the visibility boundary of the robot is an approximation and not the actual visibility boundary. The actual visibility boundary ends a small distance δ behind the visible boundary (Figure 4.6.1). As a result, though robots seem to be within the visibility boundary to a human observer it might not be so.

4. **Agreements**

   Agreements and Initial Conditions cannot be setup together. Initial Conditions should be applied before placing any robot in the Universe, agreements can be applied only after all the robots have been placed, once the agreement is applied the robot positions are altered.

5. **Sycamore Packaging**

   The current version of Sycamore is accidentally packed with some plugins, as a result if a plugin with the same name as one in the package is included, Sycamore will choose the in-built plugin instead of the new plugin.

6. **Floating Point Imprecision**

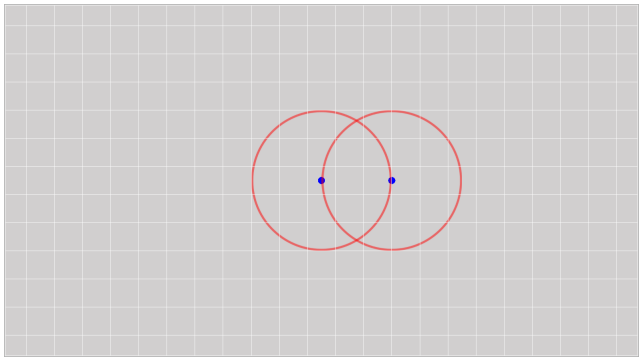


Figure 4.6.1: Robots placed V distance apart do not see each other

Sycamore uses float values to denote positions, floats have a very low precision compared to double and big decimal. The imprecision causes computation which require higher degree of precision to fail. A simple and practical example to illustrate the effect is, if the visibility radius $V = 5$ and two robots are placed in the universe at positions (0, 0) and (5, 0), the robots will not see each other due to the imprecision.

7. **Plugins SDK**

The Sycamore Plugins SDK is outdated, it does not have the current interfaces supported by Sycamore, as a result an implemented plugin might not work as expected.

8. **Visibility Graph produces predictable crowd**

The plugin provided by Sycamore to obtain a Connected Initial Visibility Graph produces predictable regions in the View Port with a denser population of robots. The property is discussed in chapter.

9. **Measure at every cycle**

This is a feature suggestion, the idea is to provide an interface to take measurements in each robot cycle for every robot.

10. **Multiple Agreements**

    This is a feature suggestion, the idea is to provide a way to allow multiple agreements, for example, there is no way at present to enable chirality and agree on distance units other than implementing a new plugin which does both.

11. **Status Messages**

    Sycamore Batch Simulator only prompts listeners when individual simulations end and when the batch finishes, it would be helpful to provide more status messages or even a progress of the simulation.

12. **Toggle Timeline**

    The Sycamore Batch Simulator turns off the Live Mode to speed up simulations, it would be useful to provide an option to toggle Live Mode.

13. **Skip Simulation**

    It would be useful to skip a single simulation in the batch, perhaps due to inactivity for a long time. This would prevent users from having to reconfigure the batch to overcome a single failed simulation, additionally if the live mode could be toggled it would be useful to review the cause of such failures.

14. **Console Output**

    Most measures print results in the system console, the system console in Sycamore Batch Simulator is not bridged to the GUI console, it would be useful to create another window to print all the messages sent to the system console.

15. **3D Simulations in Batch**

    Sycamore Batch Simulator only supports 2D plugins at present, extending to 3D is a necessary feature to make full use of Sycamore and its plugins.

16. **Extension library**

The extension library provided with Sycamore Batch Simulator is scarcely used at present, it can be extended to add features which may be toggled through the GUI.

17. **Settings Retention**

    It will be useful to preserve the simulation settings in Sycamore Batch Simulator to a file similar to Sycamore when the simulator is closed and reload the settings when the simulator is opened. This is useful when multiple batches are performed with little difference between each, for example varying the visibility radius by 1 unit in each batch.

18. **Sycamore**

    Sycamore is focused on visualization which is useful but most simulators are not used only for visualization. The simulator should be restructured such that GUI and visualization is an extension to the actual simulator, this allows extending the simulator for other purposes without many overheads. Some examples we encountered while developing the batch simulator are, the JME is tied to the engine such that lights are not possible in the batch simulator, applying visibility needed a complex setup sacrificing performance and JME threads overlapping required a cool down time.

19. **Networked simulations**

    Restructuring the simulator causing GUI and visualization to become extensions will allow creating a Network based GUI, which would let users make use of local area networks or even the internet and perform batch simulations distributed over multiple computers, this will be useful to perform simulations in the order of millions.

Chapter 5

Exact Gathering

In this chapter we study the point convergence algorithm of *Ando et al.* described in [3]. Our goal is to study the robots' behavior by observing their movements in presence of crashed robots that do not move throughout the execution. The healthy robots can not differentiate between other healthy robots and faults. The experiments are performed with three classes of initial configurations explained in the experiment setup (Section 5.3). We observe one of three behaviors: oscillation, pattern formation and point formation. These behaviors are discussed in Section 5.4 and Section 5.5.

5.1 Algorithm

The classical point convergence algorithm of [3] is designed to work with a semi-synchronous scheduler and guarantees convergence to a point. The algorithm requires that every robot in the configuration is healthy, the visibility graph of the initial configuration is connected and each robot is capable of executing its look-compute-move-sleep life-cycle.

In the algorithm, robot r_i determines the smallest enclosing circle $\mathcal{C}_i$ of all the robots in the obtained snapshot S_i including itself. The robot moves Δ distance toward the center c_i of the smallest enclosing circle, the distance Δ is computed in such a way that the farthest distance travelled by any robot r_j in a different direction would maintain its

visibility with r_i.

Algorithm 5.1 Distributed Memory less Point Convergence Algorithm for Mobile Robots with Limited Visibility

1: $S_i \subseteq \{r_j \,|\, dist\,(r_i,\, r_j \leq V)\}$
2: $\mathcal{C}_i \leftarrow$shortest enclosing circle of all robots in S_i including r_i
3: $c_i \leftarrow center\,(\mathcal{C}_i)$
4: $GOAL \leftarrow dist\,(r_i,\, c_i)$
5: **for all** $r_j \in S_i$ and $r_j \neq r_i$ **do**
6: $\quad d_j \leftarrow dist\,(r_i,\, r_j)$
7: $\quad \theta_j \leftarrow \widehat{(c_i)\,(r_i)\,(r_j)}$
8: $\quad l_j \leftarrow \frac{d_j}{2}\cos\theta_j + \sqrt{\left(\frac{V}{2}\right)^2 - \left(\frac{d_j}{2}\sin\theta_j\right)^2}$
9: **end for**
10: $LIMIT \leftarrow min\,(l_j)$
11: $\Delta \leftarrow min\,(LIMIT,\, GOAL)$
12: $Destination \leftarrow$point on the line $\overline{(r_i,\, c_i)}$ Δ distance away from r_i

The algorithm guarantees convergence in semi-synchronous scheduler but we know that the algorithm causes exact gathering in a fully synchronous scheduler.

5.2 Exact Gathering in presence of Crashed Robots

We only study the impact of crashed faults therefore the term fault always refers to a crash fault. The algorithm does not natively support working with faults, the robots executing the algorithm have no way of evaluating the nature of a robot. In fact, the obliviousness of robots will prevent them from learning that a robot is stationary through many look-compute-move-sleep cycles.

The inability to identify faults causes the robots to exhibit new behaviors other than plain convergence. We see that the healthy robots perform one of three actions namely: point formation, pattern formation or oscillation. Point formation is when the robots gather to a single point. Pattern formation is when the robots stop moving because they have satisfied the algorithm but they do not all reach a single point because of the faults (the patterns thus formed are referred to as *stable patterns*). An oscillation is

when the robots constantly switch between two patterns, for example, $\mathcal{A}(\mathbb{P}_1) = \mathbb{P}_2$ and $\mathcal{A}(\mathbb{P}_2) = \mathbb{P}_1$ where $\mathcal{A}$ is the algorithm as a function over pattern $\mathbb{P}_1$ and pattern $\mathbb{P}_2$.

5.3 Experiment Setup

We have implemented the algorithm in Sycamore, the 2D - 3D simulation engine. All our experiments are performed with a fully synchronous scheduler in Sycamore2D. We perform experiments in 2D and 1D, experiments in 1D are implemented by placing the robots at positions $y = 0$. The size of the configuration for experiments in 1D varies depending on the experiment whereas for experiments in 2D size of the configuration for each experiment is $n = 100$, the size is selected with compliance with Sycamore's first robot list limit[1]. The number of crashed robots is $f > 1$, clearly if there is single crashed robot the other robots will eventually join it. The robots are placed such that there is not one crashed robot that is not seen by a healthy robot. The possible initial configurations can be classified as follows:

1. **Crashed Robots in Proximity**

   The distance between two crashed robots is $\mathcal{D} \leq 2V$.

2. **Crashed Robots not in proximity on Vertices of a Regular Polygon (for $f \geq 3$ robots)**

   The smallest distance between any two robots is $> 2V$. Moreover, the robots are placed on the vertices of a regular polygon. A polygon is a 2D object and cannot be constructed on a line.

3. **Crashed Robots not in Proximity**

   The crashed robots do not form a regular polygon, and there exists at least a pair of crashed robots which are $\mathcal{D} > 2V$ distance away from each other.

[1]Robot list is term specific to Sycamore, it is a set of robots executing the same algorithm

The three classes are mutually exclusive and include all possible initial configurations.

5.4 Robot Behavior in 1D

We performed experiments with numerous crashed robots and healthy robots placed at random positions in 1D, we decided to study specific behaviors with only two crashed robots. We observe either point formation or pattern formation in 1D. In 1D we observe a reducing behavior where the distance travelled by each robot in a cycle is smaller than the distance it travelled in its previous cycle, hence oscillation cannot take place.

5.4.1 Simplified Algorithm

The point convergence algorithm (Algorithm 5.1) in 1D may be simplified as shown in Algorithm 5.2. Take note that center of the smallest enclosing circle in 1D is the position at the middle of the two most distant robots in the snapshot. It is easy to see that robots will not lose visibility in a fully synchronous scheduler following the simplified algorithm.

Algorithm 5.2 Simplified Point Convergence for 1D

Let r_i be the robot executing the algorithm and r_j be a robot on its left and $r_{j'}$ a robot on its right, let d_j be its distance from r_j and $d_{j'}$ bet its distance from $r_{j'}$.

- If there is no robot besides r_i then the algorithm terminates
- If there is no robot $r_{j'}$ but there exists r_j then r_i moves towards r_j and travels $\frac{d_j}{2}$ distance
- If there is no robot r_j but there exists $r_{j'}$ then r_i moves towards $r_{j'}$ and travels $\frac{d_{j'}}{2}$ distance
- If robots r_j and $r_{j'}$ both exist then r_i moves to the middle of r_j and $r_{j'}$ such that $dist\,(r_j,\, r_i) = dist\,(r_i,\, r_{j'}) = \frac{d_j + d_{j'}}{2}$.

5.4.2 Two Crashed Robots in Proximity

This case is when the distance between the two crashed robots is $\leq 2V$. In obedience to Algorithm 5.2, the healthy robots move toward the center of the line in the following fashion.

1. In the final pattern, the line is always delimited by the crashed robots. A healthy robot, if one lies at a position beyond a crashed robot outside the line, it moves inward towards the center of the line.

2. The center of the smallest enclosing circle once all the robots enter the line limited by the crashed robots is unambiguous because both the crashed robots are visible and no robot lies beyond them.

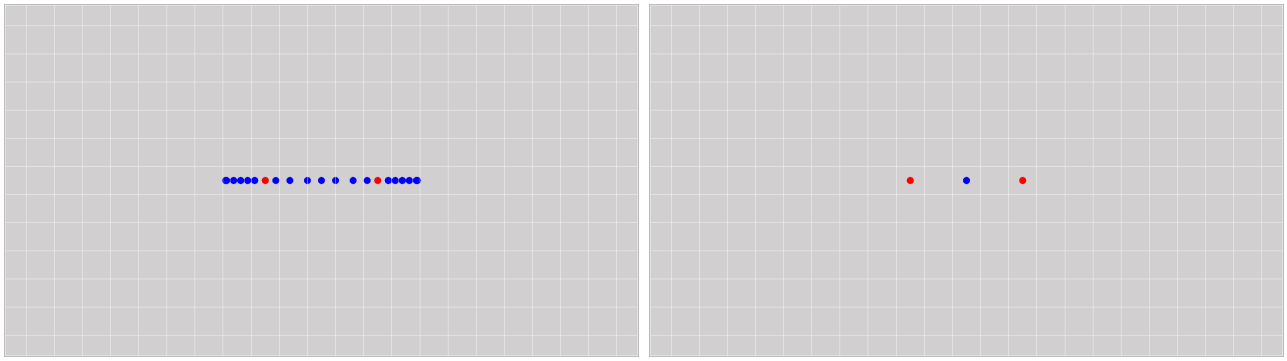
3. Finally, the healthy robots rest at a position midway between the crashed robots resulting in a point formation.

5.4.3 Two Crashed Robots not in Proximity

This case is when every robot lies on a straight line where exactly two crashed robots, f_1 and f_2, lie at a distance $> 2V$ from each other. The robots in this configuration always form a stable pattern. One possible stable pattern is a sequence of robots at regular intervals from f_1 till f_2, this pattern is called *sequential stable pattern*. Below are some observations derived from the experiments.

1. **The line occupied by the robots constantly shrinks till the extremes are occupied by the crashed robots.**

   A healthy robot, if one lies at a position outside the line (beyond a crashed robot) to the left or the right, moves inward towards the center of the line.



(a) Robots moving towards the center (b) Robots at the center when $dist\,(f_1,\, f_2) \leq 2V$

Figure 5.4.1: Line constantly shrinks

2. **Distance between robots**

   If there lies a robot r_j to the left of r_i and $dist\,(r_j,\, r_i) = x \leq V$ then there lies a robot $r_{j'}$ to the right of r_i and $dist\,(r_i,\, r_{j'}) = d;\ x \leq d \leq V$.

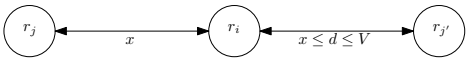
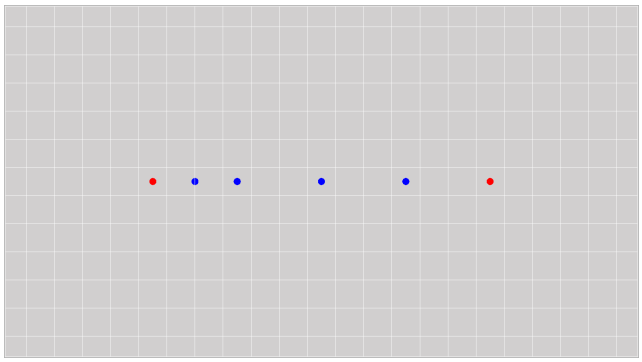


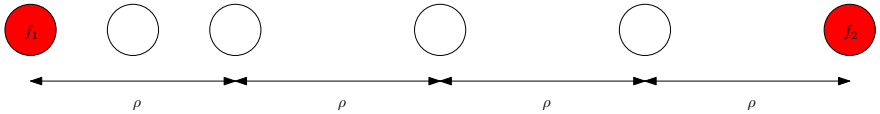
Figure 5.4.2: Robot's position and balance

3. **Intervals in Stable Patterns**

   In every stable pattern there exists a distance ρ such that from f_1 till f_2 there lies at least one robot every ρ distance



(a) Simple example to show intervals in stable patterns

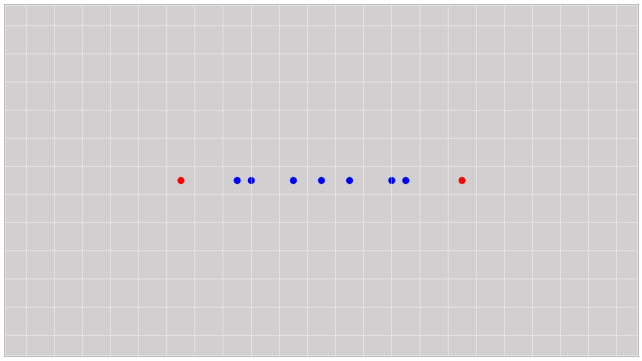


(b) Illustration to explain the intervals in stable patterns

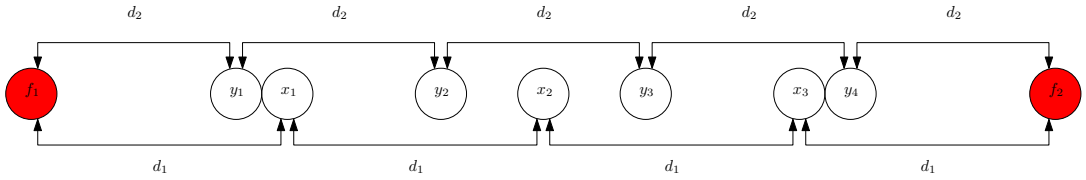
Figure 5.4.3: Intervals in Stable Patterns

4. **Bi-sequential stable pattern**

It is possible to have two sequences running in the same stable pattern, referred to as bi-sequential stable pattern. For example, the smallest inter-robot space in Figure 5.4.4 is $dist\,(y_1,\, x_1) = dist\,(x_3,\, y_4) = \frac{V}{5}$.



(a)



(b)

Figure 5.4.4: Bi-sequential Stable Pattern

5. The smallest interval in a sequential stable pattern is greater than $\frac{V}{2}$.

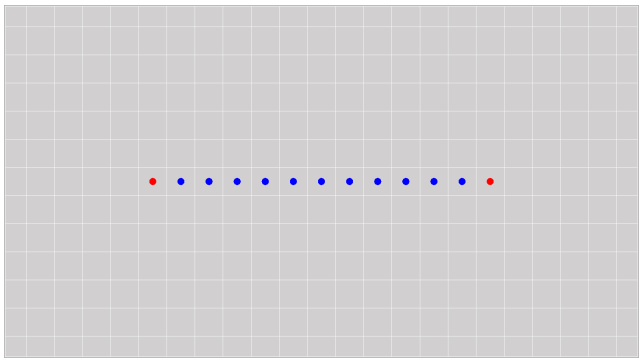


Figure 5.4.5: Smallest interval in a sequential stable pattern

6. **Robots at specific positions**

   It is possible to construct a stable pattern where chosen positions are occupied by one or more robots. The easiest method to achieve this is to use the sequential stable pattern. If the desired positions to place the robots were $(p_1,\, 0)\,,\ (p_2,\, 0)\ldots(p_n,\, 0)$, then place the robots at an interval $I = GCD\,(p_1,\, p_2\ldots p_n)$. Robot f_1 which is the left crashed robot will be placed at $(p_1 - I,\, 0)$ and the right crashed robot f_2 will be placed at $(p_n + I,\, 0)$. We also know from an earlier property that $V < 2I$.

   In the example shown in Figure 5.4.6 the selected locations $(2,\, 0)\,,\ (4,\, 0)\,,\ (8,\, 0)$ and $(14,\, 0)$ are marked in green and the crashed robots f_1 and f_2 are marked in red. The interval I is $GCD\,(2,\, 4,\, 8,\, 14) = 2$ and the selected visibility radius is $V = 3$.

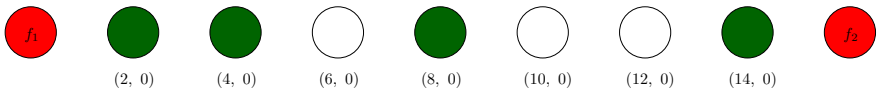


Figure 5.4.6: Robots at specific positions

7. **Sequence of Symmetric stable pattern**

   It is possible to form a chain of repeating stable patterns and the entire configuration to remain stable. The screen shot in Figure 5.4.7a is repetition of the stable

pattern shown in Figure 5.4.7b twice with three robots in between the repeated patterns to stabilize the entire pattern.

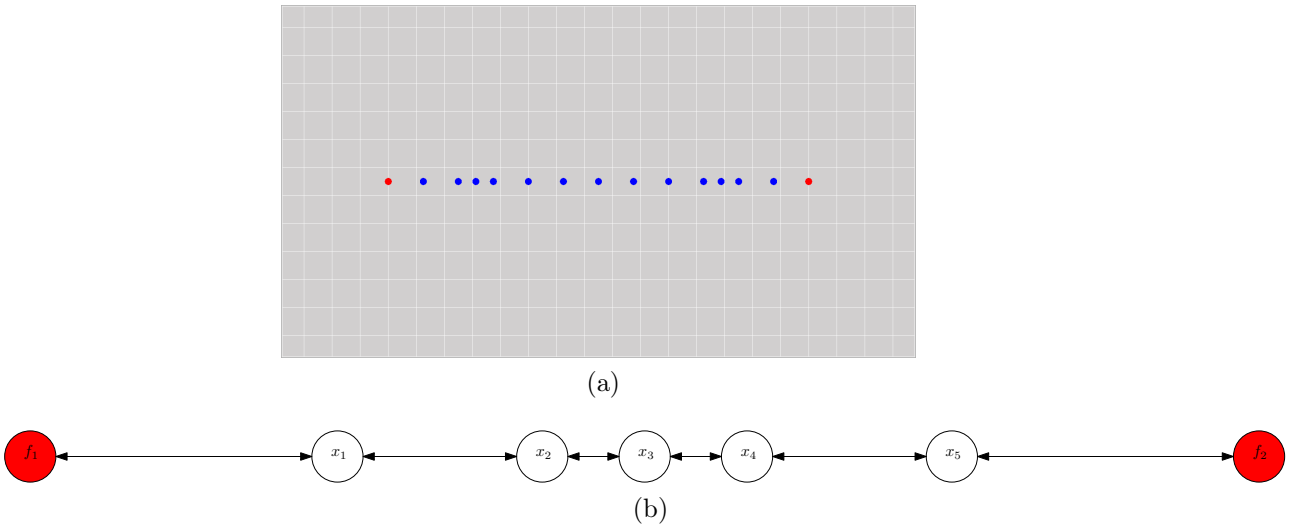


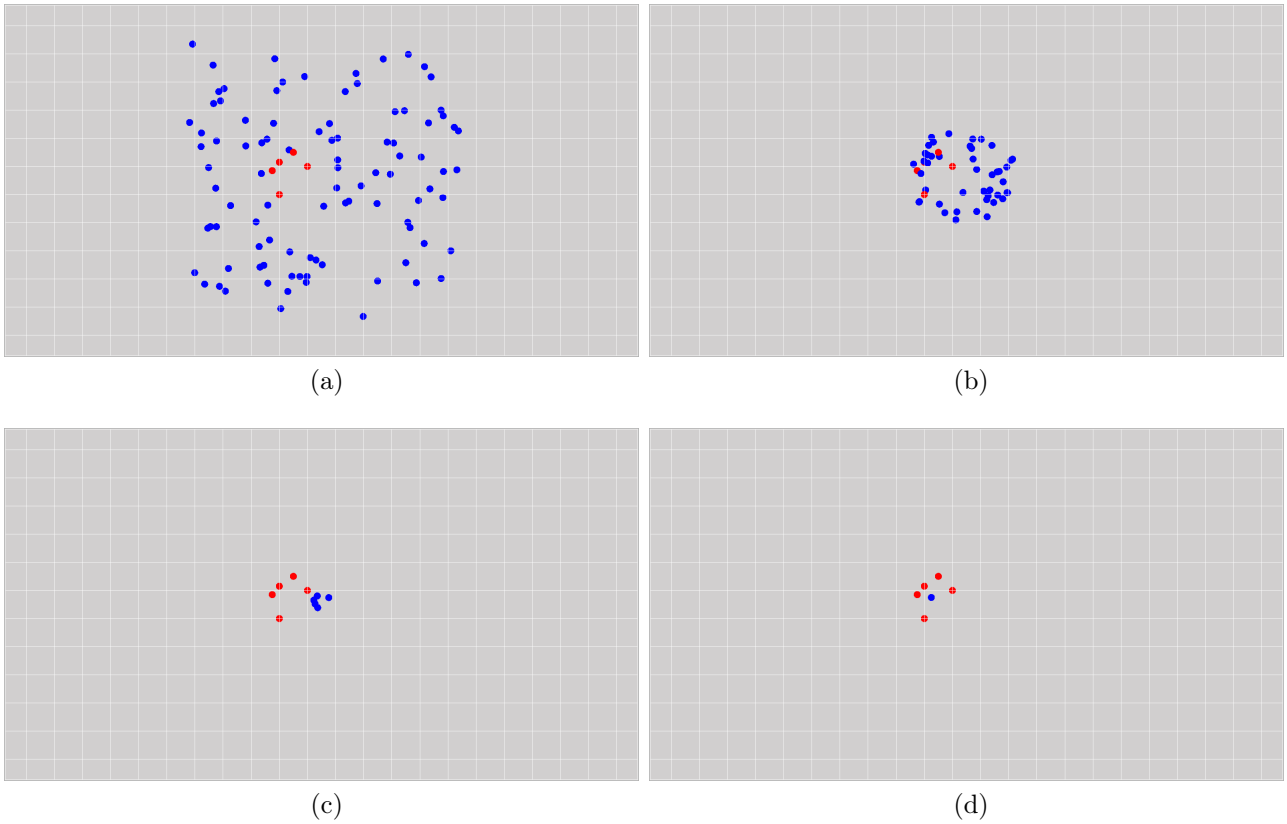
Figure 5.4.7: Sequence of Symmetric stable pattern

5.5 2D Setup

We placed numerous crashed robots and healthy robots at random positions in 2D, in this Section we have recorded our observations on the robots' behaviors when the crashed robots are in proximity and not in proximity. We have studied a special case when the crashed robots are not in proximity, and they are placed at the vertices of a polygon.

5.5.1 Crashed Robots in Proximity

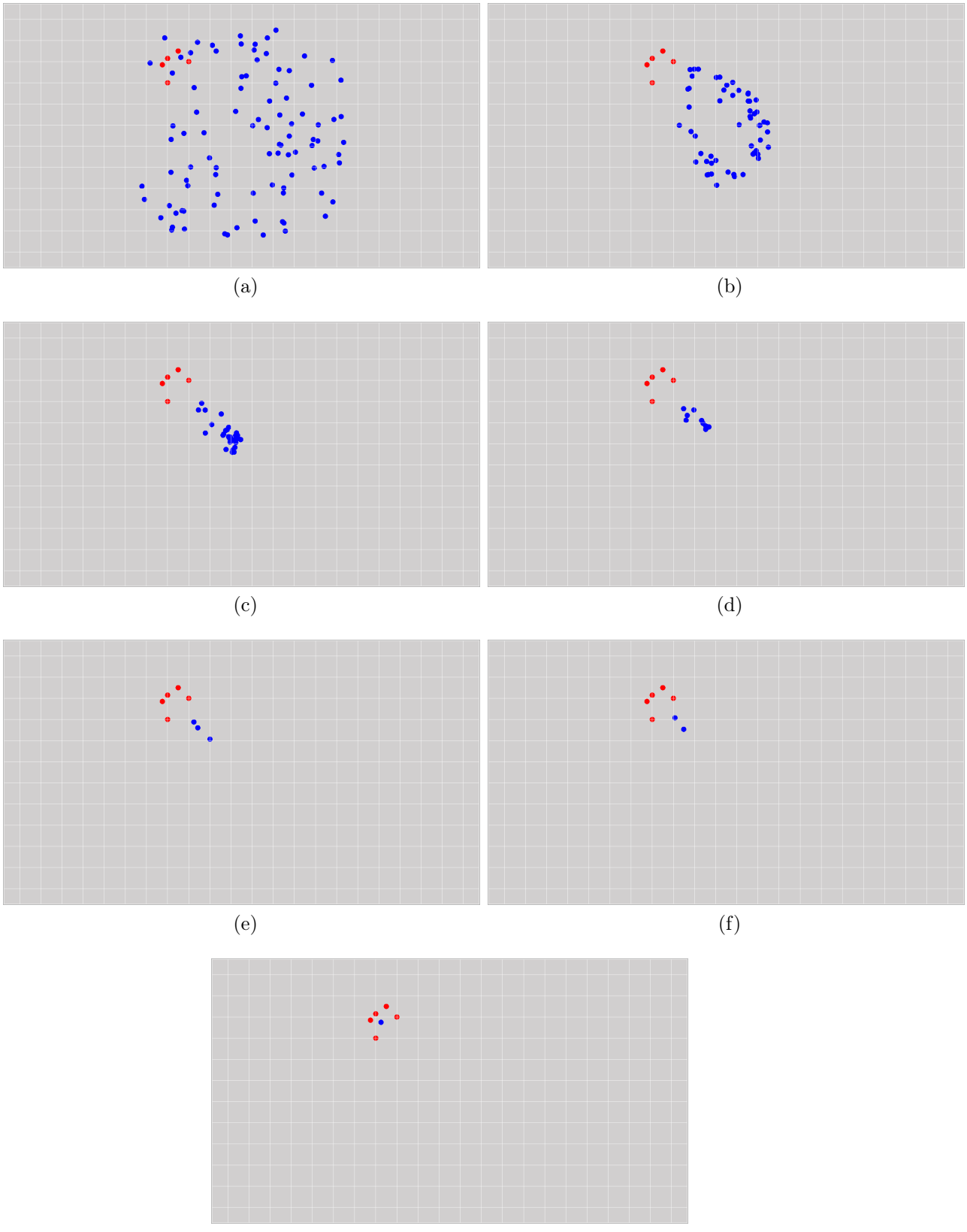
Crashed robots are in proximity if the radius of the smallest enclosing circle of all crashed robots is $\leq 2V$. At the first iteration of the robot's life-cycle every robot moves toward the center of the visibility graph. If the center c_i of the smallest enclosing circle of all the crashed robots is approximately at the center of the visibility graph, the healthy robots move close to the original center and then move together to c_i as shown in Figure 5.5.1. If c_i lies at the boundaries of the visibility graph we observe the healthy robots form a line pointing towards c_i and then gather at c_i as shown in Figure 5.5.2.



(a) (b)

(c) (d)

Figure 5.5.1: Crashed Robots in Proximity and close to the center



(a) (b) (c) (d) (e) (f) (g)

Figure 5.5.2: Crashed Robots in Proximity and close to the border

5.5.2 Crashed Robots not in proximity and on vertices of a Regular Polygon

Crashed robots placed on vertices of a polygon is a special case of crashed robots not in proximity. We observed oscillations in our experiments with random positions when crashed robots formed a pattern close to a regular polygon, we selected this special case initially to know whether this behavior is caused by imprecisions in Sycamore or by the nature of the algorithm. The configuration is simple, the healthy robots are randomly placed in a connected initial visibility graph and the crashed robots are placed at positions representing vertices of a regular polygon (Figure 5.5.4) whose center is the center of the visibility graph formed by the healthy robots. The adjacent distance between every crashed robot (in a clock wise notion) is the same, and it is $> 2V$. It is interesting to note that the point convergence algorithm performing exact gathering without crashes typically forms a polygon before completing its execution (Figure 5.5.3).

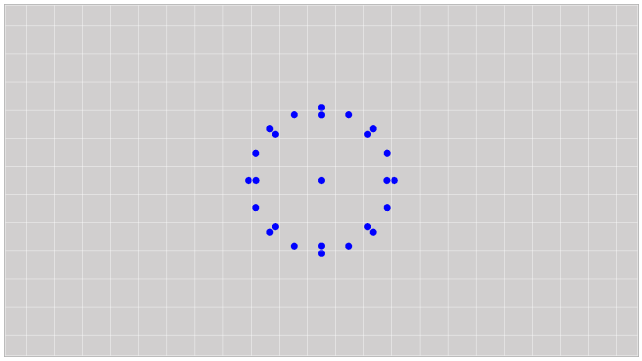
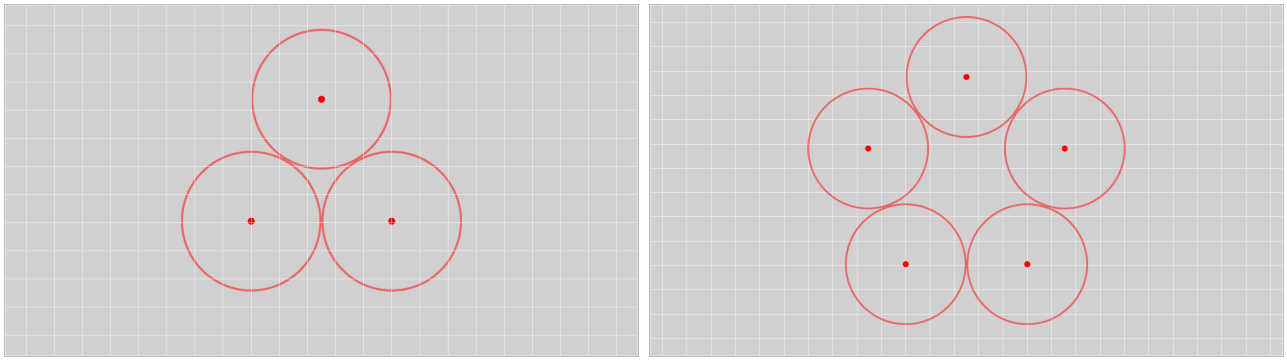


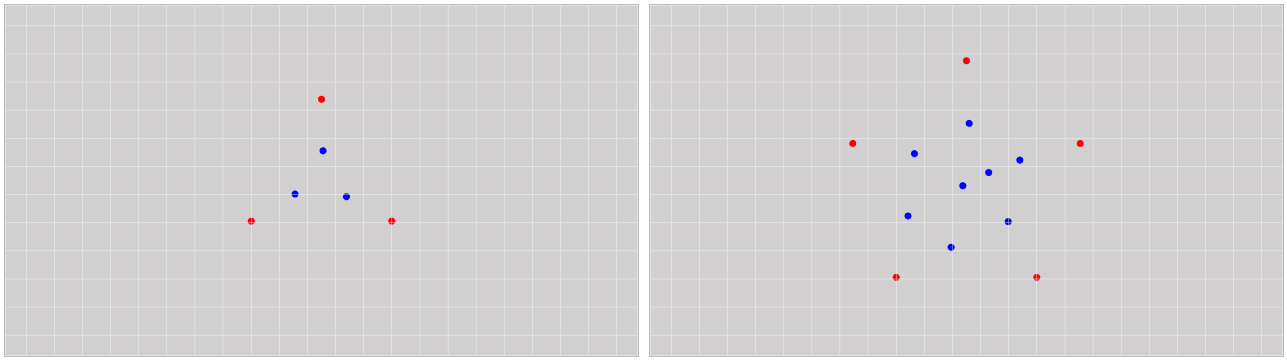
Figure 5.5.3: Robots form a circle



(a) Crashed Robots on Vertices of a Triangle (b) Crashed Robots on Vertices of a Pentagon

Figure 5.5.4: Crashed Robots on Vertices of a Polygon

In polygons with fewer crashed robots, like the triangle and pentagon, we see that the robots move towards the center and form a pattern closely resembling the polygon (Figure 5.5.5). With more crashes on the polygon, like the hexagon, the healthy robots form a pattern which better resembles a circle than a polygon (Figure 5.5.6), this is because, when the distance between adjacent vertices of the polygon remains constant and the number of vertices increase, the size of the polygon increases.



(a) Stable pattern with crashed robots on vertices of a triangle (b) Stable pattern with crashed robots on vertices of a pentagon

Figure 5.5.5: Stable pattern with crashed robots on vertices of a polygon

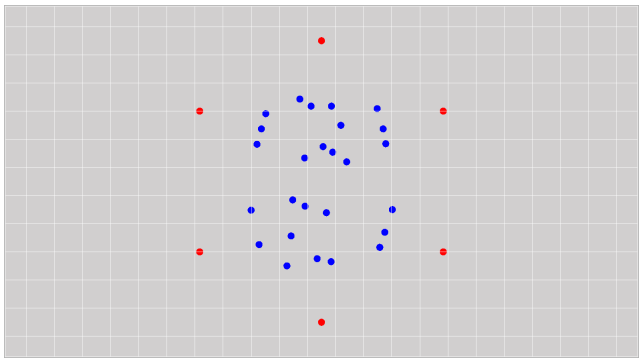
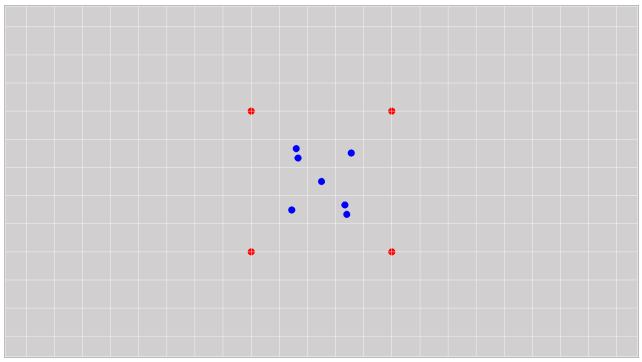
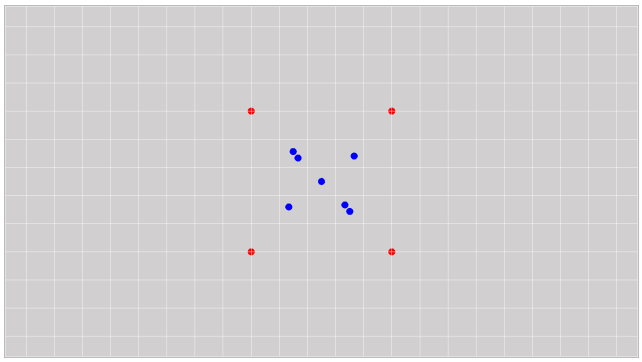


Figure 5.5.6: Stable pattern with crashed robots on vertices of a hexagon

The square, though a simple polygon, is observed to be a special case. It is in a square that we first observed the oscillation behavior, the robots within the square often oscillate such that $\mathcal{A}(\mathbb{P}_1) = \mathbb{P}_2$ and $\mathcal{A}(\mathbb{P}_2) = \mathbb{P}_1$ where $\mathcal{A}$ is the algorithm as a function over pattern $\mathbb{P}_1$ and pattern $\mathbb{P}_2$. The occurrence of an oscillation is determined by the initial configuration of the robots, when the robots are placed at random positions in a square formed by the crashed robots we often observe oscillations (e.g. see Figure 5.5.7). We hypothesize this behavior is due to the higher degree of symmetry in a small sized polygon like the square, an octagon has a higher degree of symmetry but the polygon is big.



(a) Pattern $\mathbb{P}_1$

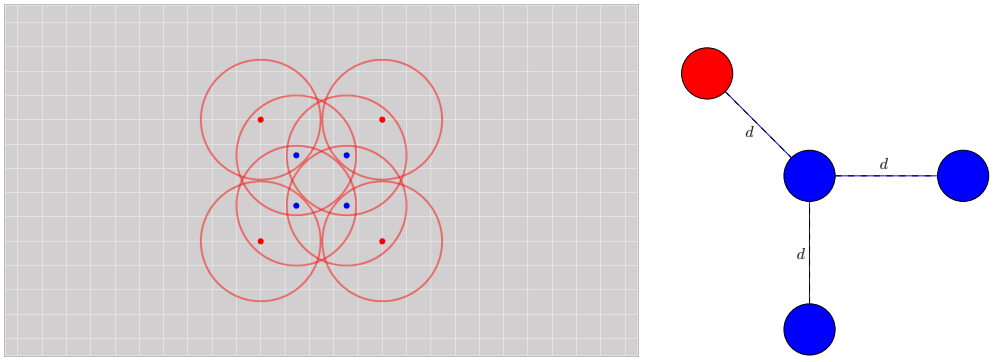


(b) Pattern $\mathbb{P}_2$

Figure 5.5.7: Oscillation pattern with crashed robots on vertices of a square

That being said it is also possible to design initial conditions which do not lead to an oscillation, Figure 5.5.8a is a simple configuration without an oscillation. Ev-

ery healthy robot in Figure 5.5.8a sees two adjacent healthy robots and one crashed robot, the distance between the observing robot and other robots is $d = 4.14$. Positions of healthy robots are $(-2.07,\ 2.07)$, $(2.07,\ 2.07)$, $(2.07,\ -2.07)$ and $(-2.07,\ -2.07)$, positions of crashed robots are $(-5,\ 5)$, $(5,\ 5)$, $(5,\ -5)$ and $(-5,\ -5)$. Figure 5.5.8 is a simple example to demonstrate a pattern formation in the square, we have also observed complex patterns as shown in Figure 5.5.9.



(a) (b)

Figure 5.5.8: Simple Pattern formation with crashed robots on vertices of a square

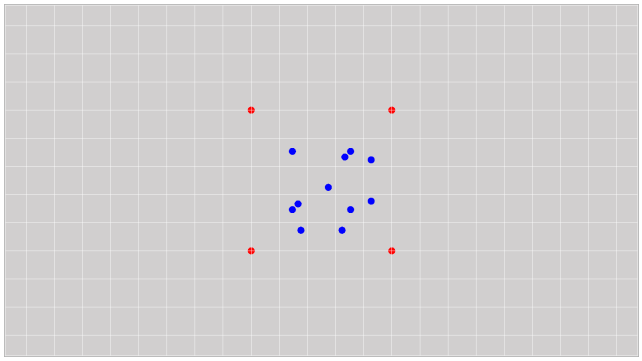


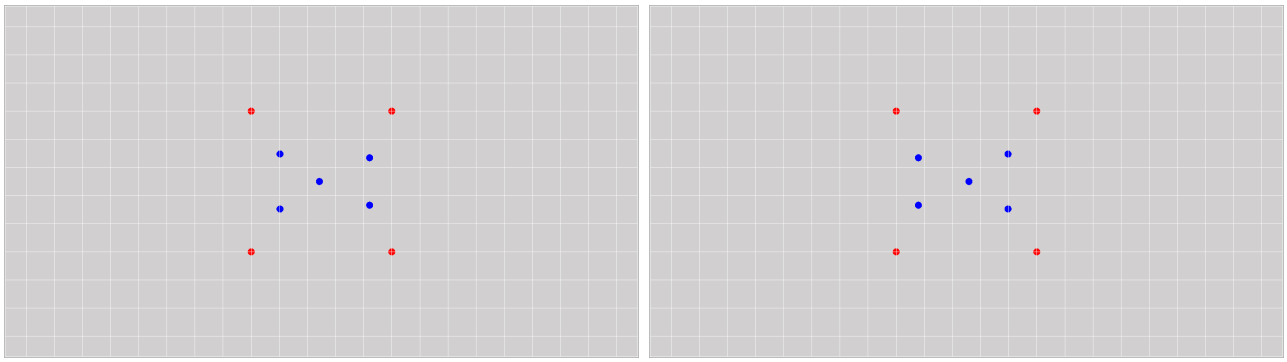
Figure 5.5.9: Complex pattern formation with crashed robots on vertices of a square

The oscillation we observe in Figure 5.5.7 is not assuring, the distance travelled by robots from pattern $\mathbb{P}_1$ to $\mathbb{P}_2$ is small, this might have been caused by the floating point imprecision in Sycamore. We formulated the algorithm and our observation of what constitutes to an oscillation, we programmed Mathematica to generate an oscillating pattern if one exists. Mathematica confirmed the existence of such oscillating patterns and calculated the following approximate coordinates;

$\mathbb{P}_1 = (-2.925854, 1.967469), (3.450008, 1.668956), (3.450008, -1.668956), (-2.925854, -1.967469)$ and $(-0.176950, 0.000000)$

$\mathbb{P}_2 = (-3.450008, 1.668956), (2.925854, 1.967469), (2.925854, -1.967469), (-3.450008, -1.668956)$ and $(0.176950, 0.000000)$

We tried the generated pattern in Sycamore and observed a cleaner oscillation, the oscillation is shown in Figure 5.5.10

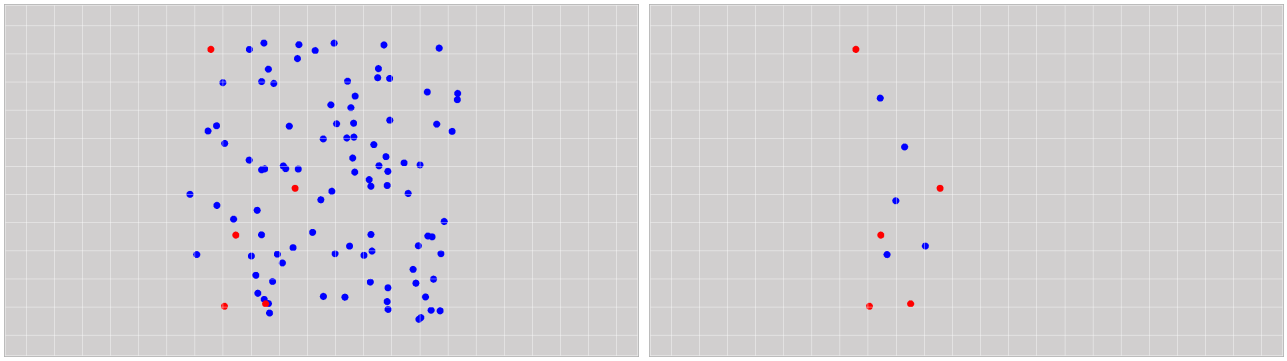


(a) (b)

Figure 5.5.10: Oscillation Pattern

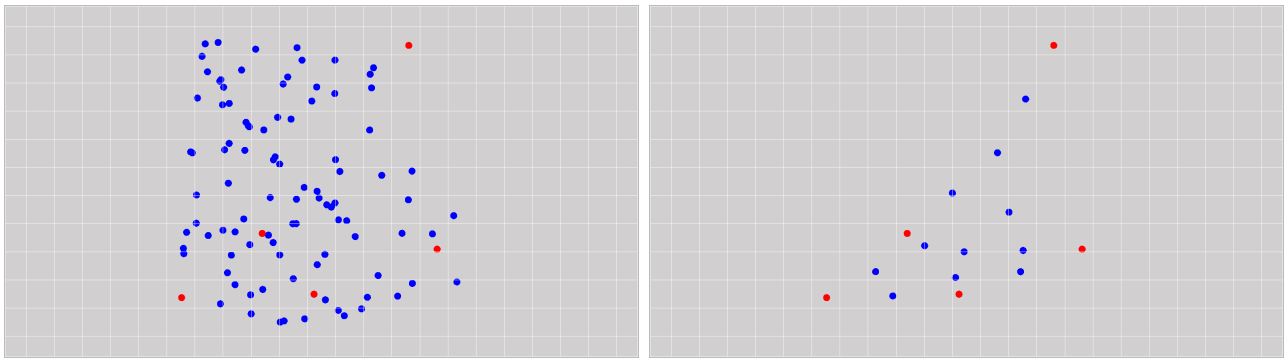
5.5.3 Crashed Robots not in Proximity

The radius of the smallest enclosing circle of all the crashed robots is $> 2V$ and these crashed robots do not take positions on a polygon. The behavior is very simple, the robots move toward the center of the visibility graph and often form a pattern. The size of the final stable pattern is always equal to the size of the smallest enclosing circle of all the crashed robots. This configuration will never lead to a point formation and often leads to a stable pattern. Oscillation is possible in this configuration but the occurrence is very minimal and we observe it rarely.



(a) (b)

Figure 5.5.11: Crashed Robots Not In Proximity - Sample A



(a) (b)

Figure 5.5.12: Crashed Robots Not In Proximity - Sample B

5.6 Conclusion

In this chapter we studied the impact of crashed robots on point convergence under fully synchronous scheduler performing exact gathering. We split our experiments into two categories, 1D and 2D. We initially placed numerous crashed robots and healthy robots at random positions in both 1D and 2D, and we decided to study only specific configurations of crashed robots, the ones that seemed to generate most interesting behaviors. For Crashed robots in 1D, we observed point formation when the crashed robots were close and pattern formation when they were further apart. The properties exhibited by healthy robots in pattern formation is recorded in Section 5.4.3. Crashed robots in 2D were placed

on vertices of a regular polygon, we observed point formation when the crashed robots were close and pattern formation when the crashed robots were further apart but not on vertices of a polygon. We observed oscillation when the crashed robots were placed on vertices of a regular polygon, we often observed oscillation in the square. We used Mathematica to derive specific positions which lead to a clean oscillation easy to observe and record.

The experiments were by no means exhaustive, but only an initial analysis of what could be expected. These are some configurations in 1D which would be interesting to study in the future. For example, it might be interesting to place multiple crashed robots at equal distance and study the behavior of healthy robots within each segment and across multiple segments, also placing multiple crashed robots at random positions but not overlapping and studying the behavior of healthy robots within each segment and across multiple segments could lead to interesting observations. These are several problems in 2D which would be interesting to study in the future. First and foremost, it is important to characterize all the stable patterns that could occur for a given configuration of faults, it would be interesting to understand the nature of oscillations and, in particular, to determine if there exist oscillations involving more than two patterns.

Chapter 6

Simple Random Gathering

In this Chapter, we investigate a probabilistic algorithm to achieve gathering by a group of robots with limited visibility. The algorithm is very simple: when activated, a robot moves to join an arbitrary robot within its visibility. Following this simple strategies clearly gathering is not guaranteed. In this Chapter we analyze the success rate of the algorithm, the impact of initial conditions on the success rate, and the behavior of robots in the presence of a single crash. The main reason for implementing this algorithm was to test some of Sycamore features. While doing that, we also discovered some interesting behaviors and made some observations.

We describe the algorithm and requirements in Section 6.1. Our experiment setup is discussed in Section 6.2 and results are divided into Section 6.3, where we analyze the initial conditions and their impact on the algorithm, and Section 6.4 where we study the impact of a crash on the algorithm.

6.1 Algorithm

Robot r_i executing the algorithm determines if there exists a robot r_j in the snapshot S_i such that the distance between r_i and r_j is at least δ. If there exists such a robot then r_i picks a random robot r_{rand} from S_i and, if the distance between r_i and r_{rand} is at least

δ then r_i moves to the position of r_{rand} else r_i does not move.

Algorithm 6.1 Random Gathering

```
1: S_i ← {r_x : dist(r_i, r_x) ≤ V}
2: if ∃r_j ∈ S_i, dist(r_i, r_j) ≥ δ then
3:     r_rand ← rand(S_i)
4:     if dist(r_i, r_rand) ≥ δ then
5:         Destination ← r_rand
6:     else
7:         Do Nothing
8:     end if
9: else
10:    Terminate
11: end if
```

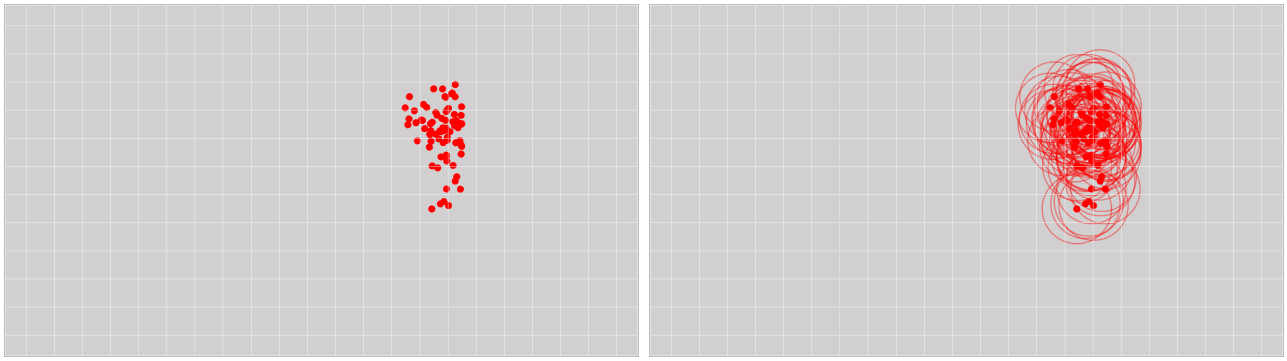
The random choice of the robots makes crowded regions favored for robot destinations. The algorithm requires circular visibility, multiplicity detection, initially connected visibility graph and healthy robots.

6.2 Experiment Setup

We consider fully synchronous executions and we start from an initially connected visibility graph. The first observation we can make is that the initial position of the robots greatly influence the result. To better understand the impact of the initial configuration, we perform the experiments with three different classes of initial configurations.

1. **Sycamore's Default Connected Visibility Graph (SCVG)**

   In an empty universe the first robot is placed at a random position in the view port. Positions to place other robots are selected by first choosing a random robot r_j from the universe and then selecting a random position in r_j's visibility circle.

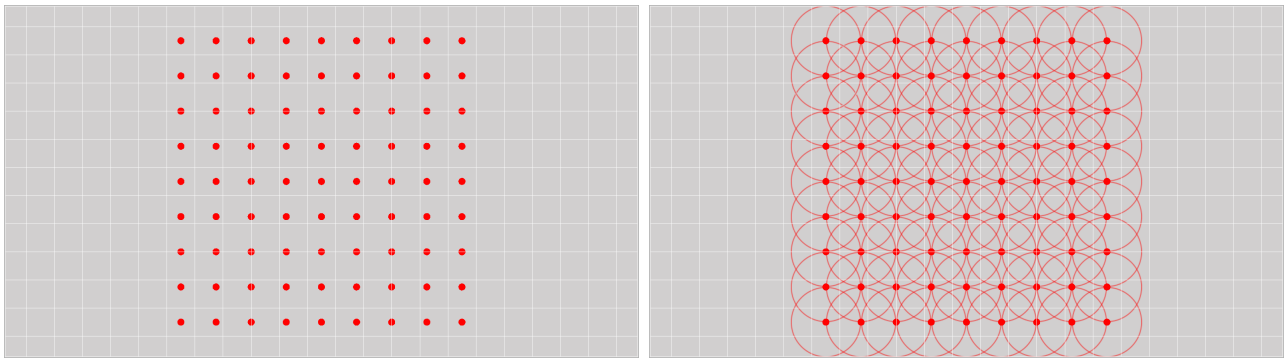


(a) Robot Positions (b) Robot Visibilities

Figure 6.2.1: Sycamore's Default Connected Visibility Graph

2. **Mesh-Like Connected Visibility Graph (MCVG)**

   The robots are first placed on a "skeleton", it is a mesh-like structure which is constructed to ensure that there is no position in the view port that is not seen by a robot in the "skeleton". The distance between rows and columns is V. Once the "skeleton" has visibility of every position in the view port, the other robots are placed at random positions inside the view port.



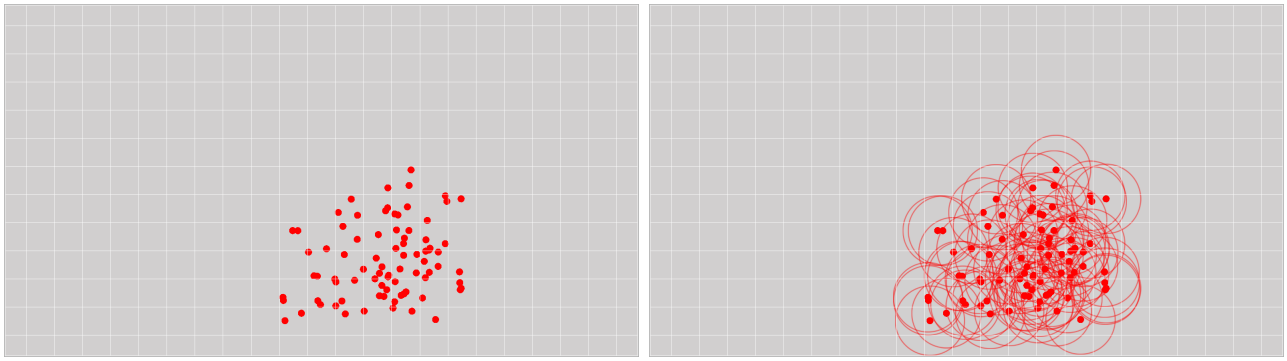
(a) Robot Positions (b) Robot Visibilities

Figure 6.2.2: Mesh-Like Connected Visibility Graph

3. **Random Connected Visibility Graph (RCVG)**

   In an empty universe the first robot is placed at a random position in the view port. The positions to place the other robots are selected at random and, it is evaluated

after each insertion, whether the visibility graph would remain connected. If the selected position breaks the visibility graph, then the position is abandoned and the selection process repeats till a suitable position is found.



(a) Robot Positions (b) Robot Visibilities

Figure 6.2.3: Random Connected Visibility Graph

We can observe that SCVG tends to form a cluster when distributing the robots. We have introduced MCVG and RCVG to force a more uniform distribution but take note that neither MCVG or RCVG is perfectly random and uniform because MCVG creates a skeleton, hence the formation is predictable therefore not random, on the other hand it is more uniform; RCVG places the first robot at random but the successive robots are closer to robots already in the universe. In any case, MCVG and RCVG avoid creating predictable clusters like SCVG.

Experiments are conducted in Sycamore with a bound view port and a varying visibility radius. The number of robot in each simulation is 100, the number is selected with compliance with Sycamore's first robot list[1].

[1]First Robot List is a term specific to Sycamore which means the set of robots executing the same algorithm.

6.3 Impact of Initial Conditions

The first experiment is performed to ponder the impacts of the three initial conditions discussed in Section 6.2. The result is collated from 23,000 simulations performed in Sycamore Batch Simulator.

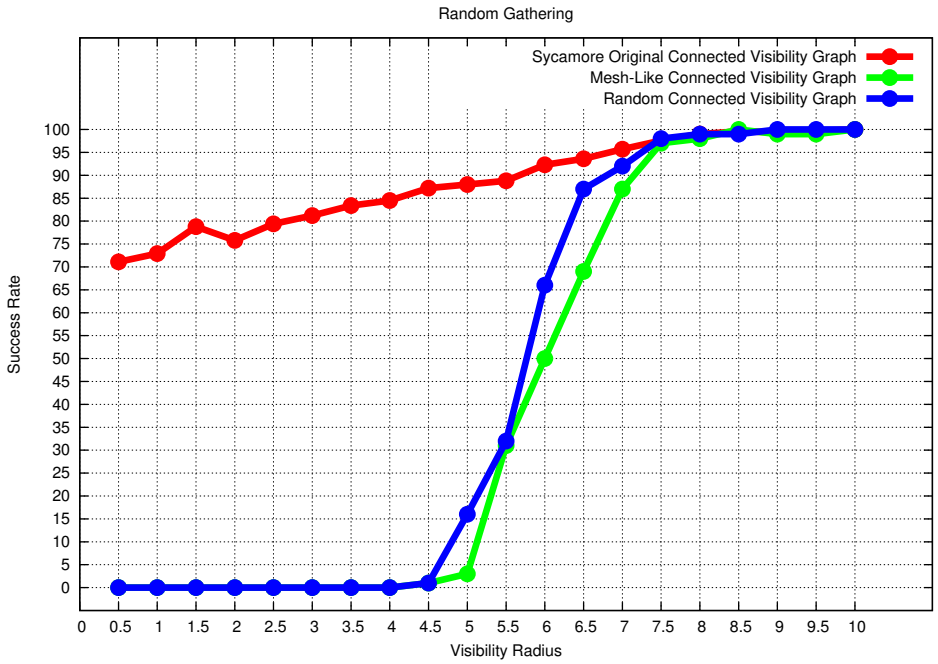


Figure 6.3.1: Random Gathering In all Initial Conditions

1. **Sycamore's Default Connected Visibility Graph (SCVG)**

   The rate of success is generally higher than the rate of success of the other initial conditions. The lowest rate of success is 71.1%, observed at visibility radius $V = 0.5$. The average rate of success over all visibilities is 88.4%. The result is higher than expected, we identify the cause of the success in the distribution of the robots at time t_0. Let us consider how the robots are placed. The first robot r_1 will be placed at a random position in the empty view port, the second robot r_2 will be placed at a random position in r_1's visibility as shown in Figure 6.3.2a. Notice that size of the view port affects r_1's position but the positions of r_2 and the other robots are not affected by it. Robot r_3 is placed by first choosing a robot r_{rand} at

random from the view port and then selecting a position at random from r_{rand}'s visibility. The chance of picking r_1 or r_2 is equal. Consider Figure 6.3.2a and say r_1 is chosen, every position in r_1's visibility has an equal chance of being selected, similarly if r_2 were chosen. Notice in Figure 6.3.2a there lies an area of visibility common to r_1 and r_2 the chance of picking a position in the common visibility area is clearly higher since it could be selected indifferently of the chosen robot. As more robots are added we observe that a cluster is formed approximately at the center of the visibility graph and r_1 lies approximately at the center of the cluster. Since a robot chooses another robot at random in its snapshot, robots are likely to choose destinations from the denser region. In fact, we observe a high rate of success because of more robots being driven to common positions.

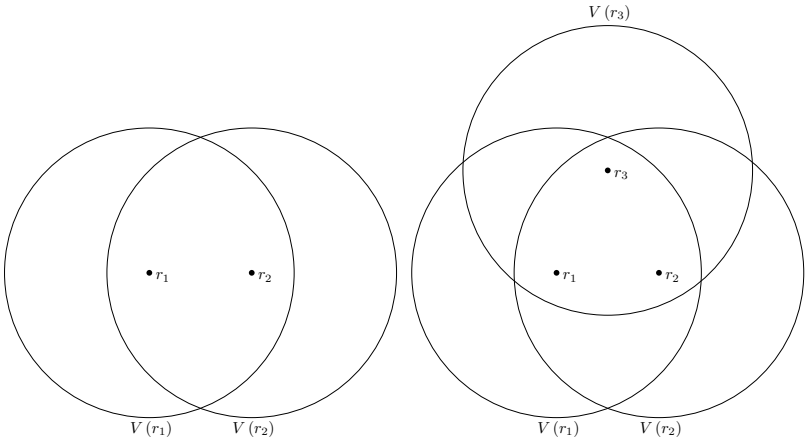


Figure 6.3.2: Robot position selection in SCVG

Visibility Diameter	Area Covered	Percentage Of ViewPort*	Gathering Success
0.5	0.786	0.196%	71.1%
1	3.143	0.786%	72.9%
1.5	7.071	1.768%	78.8%
2	12.571	3.143%	75.8%
2.5	19.643	4.911%	79.4%
3	28.286	7.071%	81.2%
3.5	38.5	9.625%	83.4%
4	50.286	12.571%	84.5%
4.5	63.643	15.911%	87.2%
5	78.571	19.643%	88%
5.5	95.071	23.768%	88.8%
6	113.143	28.286%	92.3%
6.5	132.786	33.196%	93.6%
7	154	38.5%	95.7%
7.5	176.786	44.196%	97.6%
8	201.143	50.286%	99%
8.5	227.071	56.768%	99.6%
9	254.571	63.643%	99.4%
9.5	283.643	70.911%	99.7%
10	314.286	78.571%	100%

*Percentage of viewport covered by each robot.

Table 6.3.1: Random Gathering on Sycamore's Default Connected Visibility Graph

2. **Mesh-Like Connected Visibility Graph (MCVG)**

   The rate of success is generally lower than the rate of success of the other initial conditions, we observe no success in gathering for visibilities $V \leq 4.5$. The average rate of success is 41.7%, the lowest rate of success for visibilities $V \geq 5$ is 1%. The skeleton constructed by MCVG eliminates the formation of clusters and covers the view port, consequently it increases uniformity in robot positions in the initial configuration. The lack of clusters cause robots to choose destinations with near equal probability, hence there is a higher chance of breaking the visibility graph.

Visibility Radius	Area Covered	Percentage Of ViewPort*	Gathering Success
4	50.286	12.571%	0%
4.5	63.643	15.911%	1%
5	78.571	19.643%	3%
5.5	95.071	23.768%	31%
6	113.143	28.286%	50%
6.5	132.786	33.196%	69%
7	154	38.5%	87%
7.5	176.786	44.196%	97%
8	201.143	50.286%	98%
8.5	227.071	56.768%	100%
9	254.571	63.643%	99%
9.5	283.643	70.911%	99%
10	314.286	78.571%	100%

*Percentage of viewport covered by each robot.

Table 6.3.2: Random Gathering on Mesh-Like Connected Visibility Graph

3. **Random Connected Visibility Graph (RCVG)**

   The rate of success is generally lower than SCVG but higher than MCVG, We observe no success in gathering for visibilities $V \leq 4.5$. The average rate of success is 44.5%, the lowest rate of success for visibilities $V \geq 5$ is 1%. RCVG eliminates the formation of clusters but does not uniformly cover the view port. Once the first robot r_1 is placed, the positions of robots r_2 and the other robots are not affected by the size of the view port. The lack in uniformity slightly increases the chance of some positions being selected over others, the effect piles up over multiple cycles causing an overall higher rate of success.

Visibility Radius	Area Covered	Percentage Of ViewPort*	Gathering Success
4	50.286	12.571%	0%
4.5	63.643	15.911%	1%
5	78.571	19.643%	16%
5.5	95.071	23.768%	32%
6	113.143	28.286%	66%
6.5	132.786	33.196%	87%
7	154	38.5%	92%
7.5	176.786	44.196%	98%
8	201.143	50.286%	99%
8.5	227.071	56.768%	99%
9	254.571	63.643%	100%
9.5	283.643	70.911%	100%
10	314.286	78.571%	100%

*Percentage of viewport covered by each robot.

Table 6.3.3: Random Gathering on Random Connected Visibility Graph

With all three initial configurations we find a general increase in the rate of success with the increase in visibility radius, the increase in rate of success is linear in SCVG whereas the same in MCVG and RCVG are exponential. The exponential growth suggests the success can be largely ascribed to the increase in visibility radius and neither to the algorithm nor to the initial configuration, linear growth in SCVG suggests the success is due to either the algorithm or the initial configuration but we understand from MCVG and RCVG that the algorithm does not enhance the rate of success greatly therefore the success rate is ascribed to the initial configuration. The rate of gathering success in all initial configurations normalizes at $V \geq 7.5$ at which point each robot sees approximately 44.2% of the view port, almost every robot sees each other.

6.4 Impact of Crash

It is easy to see that more than one crashed robot will deteriorate the rate of success or, even worse, it will cause the algorithm to not terminate making the robots move perpetually between positions occupied by crashed robots. In this section we consider

only a particular situation: a single crash whose position is chosen trying to favor the success rate of the algorithm. It is clear that placing the crash at the center of the visibility graph favors the success rate.

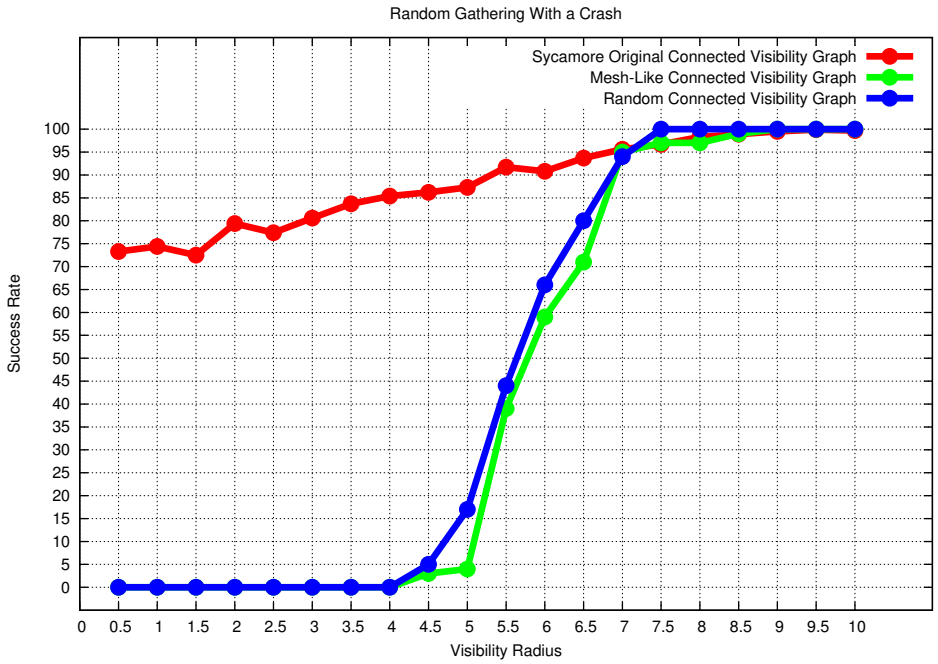


Figure 6.4.1: Random Gathering on all Initial Conditions with Fault

1. **Sycamore's Default Connected Visibility Graph (SCVG)**

   SCVG has a well defined cluster and we know the approximate position of the center of the cluster is the first robot placed in the empty view port. The crashed robot is placed first, the other robots with a high probability surround it forming a cluster. Figure 6.4.1 shows a generally higher rate of success than other initial configurations but equal or slightly lower rate of success than SCVG without fault. The higher rate of success over other initial configurations is achieved due to the same reason discussed in Section 6.3. The insignificant change to its variant without fault is because the position of the crash, the position does not negatively impact the algorithm but does not improve conditions from before since the cluster existed even without the crashed robot and other robots naturally favored the center.

Visibility Diameter	Area Covered	Percentage Of ViewPort*	Gathering Success
0.5	0.786	0.196%	73.3%
1	3.143	0.786%	74.4%
1.5	7.071	1.768%	72.5%
2	12.571	3.143%	79.4%
2.5	19.643	4.911%	77.4%
3	28.286	7.071%	80.6%
3.5	38.5	9.625%	83.7%
4	50.286	12.571%	85.4%
4.5	63.643	15.911%	86.2%
5	78.571	19.643%	87.3%
5.5	95.071	23.768%	91.7%
6	113.143	28.286%	90.8%
6.5	132.786	33.196%	93.7%
7	154	38.5%	95.6%
7.5	176.786	44.196%	96.7%
8	201.143	50.286%	98.3%
8.5	227.071	56.768%	98.9%
9	254.571	63.643%	99.5%
9.5	283.643	70.911%	99.9%
10	314.286	78.571%	99.7%

*Percentage of viewport covered by each robot.

Table 6.4.1: Random Gathering on Sycamore's Default Connected Visibility Graph

2. **Mesh-Like Connected Visibility Graph (MCVG)**

   The skeleton in MCVG covers the entire view port, hence the position which is almost equally accessible from any where in the view port is its center, the crash is placed at the center. The results we observe in relation to other initial conditions are similar to the MCVG without crash, there is an observable increase in the rate of success and more visibilities yield 100% success rates. The average rate of success over all visibilities is 43.2%

Visibility Radius	Area Covered	Percentage Of ViewPort*	Gathering Success
4	50.286	12.571%	0%
4.5	63.643	15.911%	3%
5	78.571	19.643%	4%
5.5	95.071	23.768%	39%
6	113.143	28.286%	59%
6.5	132.786	33.196%	71%
7	154	38.5%	95%
7.5	176.786	44.196%	97%
8	201.143	50.286%	97%
8.5	227.071	56.768%	99%
9	254.571	63.643%	100%
9.5	283.643	70.911%	100%
10	314.286	78.571%	100%

*Percentage of viewport covered by each robot.

Table 6.4.2: Random Gathering on Mesh-Like Connected Visibility Graph

3. **Random Connected Visibility Graph (RCVG)**

   RCVG does not have any preferred position since the robots are placed at random. The crashed robot is constantly placed at the center of the view port whereas the other robots are placed in accordance to RCVG at random positions. The average rate of success is observed to be 45.3% which is a moderate increase from when there are no crashes but it is significant to note that with a crash RCVG has yielded 100% rate of success for numerous visibilities.

Visibility Radius	Area Covered	Percentage Of ViewPort*	Gathering Success
4	50.286	12.571%	0%
4.5	63.643	15.911%	5%
5	78.571	19.643%	17%
5.5	95.071	23.768%	44%
6	113.143	28.286%	66%
6.5	132.786	33.196%	80%
7	154	38.5%	94%
7.5	176.786	44.196%	100%
8	201.143	50.286%	100%
8.5	227.071	56.768%	100%
9	254.571	63.643%	100%
9.5	283.643	70.911%	100%
10	314.286	78.571%	100%

*Percentage of viewport covered by each robot.

Table 6.4.3: Random Gathering on Random Connected Visibility Graph

6.5 Conclusion

We developed a new and simple algorithm for synchronous robots with limited visibility: *Random Gathering*. In our experiments with Sycamore we observed higher than expected success rate and identified the cause to be the connected initial visibility graph plugin from Sycamore (SCVG). We developed two new connected initial visibility graph plugins, MCVG & RCVG, and confirmed our hypothesis. We repeated the experiments on all three plugins with a single well placed crash. The results with crash on SCVG was not significantly different from results without crash. MCVG and RCVG exhibit an increase in the rate of success compared to their variants without crash, we observe more visibility radii yielding 100% rate of success with crash. We conclude that SCVG is vulnerable and could be exploited but it is the most resource friendly connected initial visibility graph plugin available for Sycamore.

These are some interesting experiments for the future. Below are some examples: The study of the rate of success if robots stop execution (crash) when they are at the position of another crash. The study of the rate of success when robots have a memory

and choose positions at random from the snapshot, then pick a random position from all the destinations they have chosen in the past including the newly selected position. The study of the rate of success with other connected initial visibility graph plugins and schedulers. The study of the rate of success if robots only choose positions that are $< V$ distance from every robot in its snapshot and characterize the nature of the algorithm in 1D and 2D.

Chapter 7

Combination Algorithm

We study the behavior of robots when the two algorithms of the previous chapters, *Random Gathering* (Chapter 6) and *Exact Gathering* (Chapter 5), are executed in the same configuration under a synchronous scheduler. In particular, we study the impact on the rate of success of *Random Gathering* in the presence of a few robots executing the *Exact Gathering* algorithm. Executing these two algorithms together in the same configuration is helpful when the system is composed of many robots with limited resources (the ones running *Random Gathering*) and only a few with more (the fewer running *Exact Gathering*). If successful, this would effectively increase the rate of success over *Random Gathering*.

The setup is defined in Section 7.1, it describes the configuration of robots executing Exact Gathering and Random Gathering. We discuss our observation in Section 7.2.

7.1 Experiment Setup

Let $\mathcal{R}$ be the set of all the robots in the configuration and $|\mathcal{R}| = 100$ is the number of robots, $\mathcal{G} \subset \mathcal{R}$ is the set of robots executing Random Gathering and $\mathcal{P} \subset \mathcal{R}$ is the set of robots executing Exact Gathering. The robots have limited circular visibility, multiplicity detection and the initial visibility graph is connected, the simulations are

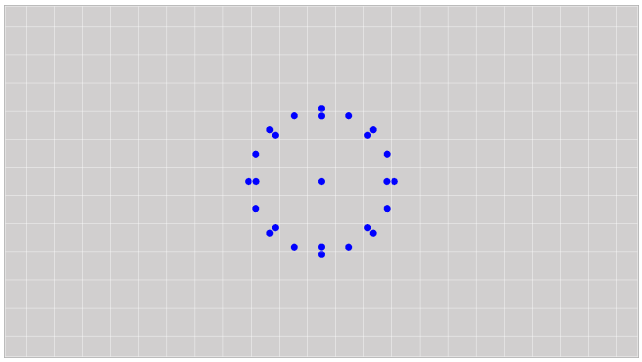
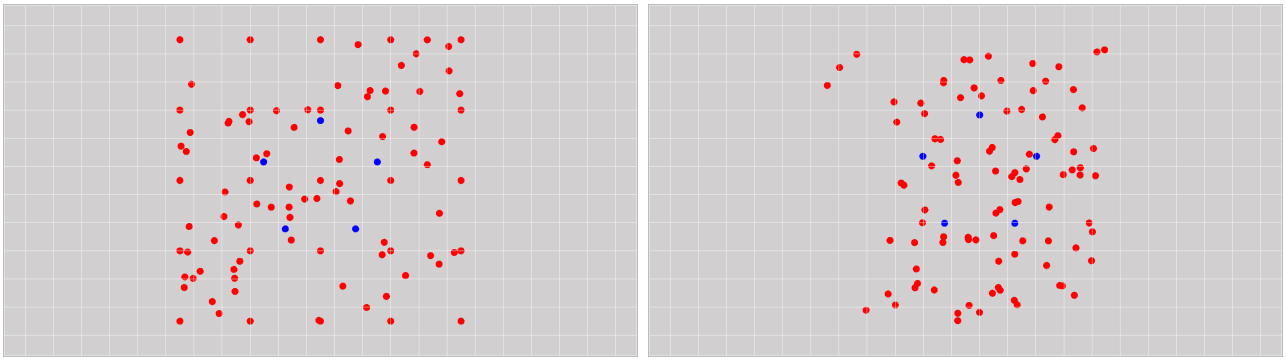


Figure 7.1.1: Pure Exact Gathering Forming an arbitrary shape

performed with a fully synchronous scheduler. The configuration is setup such that each robot executes its own algorithm without knowing what algorithm the other robots are executing.

We performed experiments placing the robots executing Exact Gathering and the ones executing Random Gathering at random positions, we observed that the rate of success was generally lower than the rate of success in Simple Random Gathering algorithm. The experiments in this chapter concern with increasing the rate of success over the Simple Random Gathering algorithm. We choose positions for robots from the observations made in the previous chapters. In a configuration of robots which execute only Exact Gathering we observe that robots form an arbitrary shape resembling a circle (Figure 7.1.1) and converge approximately at the middle of the initial visibility graph. In a configuration of robots which execute only Random Gathering we observe that robots have a higher rate of success when they are driven toward a common position or region. We expect to observe a higher rate of success than Simple Random Gathering if we position the robots executing Exact Gathering on vertices of a polygon around the center of the visibility graph. The experiments are performed on MCVG and RCVG, a sample configuration is shown in Figure 7.1.2. We do not experiment on SCVG since it natively has a common position or region towards which the robots are driven. The radius of visibility is set to a constant $V = 10$ since we seek to study only impact of the interaction, we perform experiments with $|\mathcal{P}| = 3, 4, \ldots, 10$.



(a) Position of the polygon in MCVG (b) Position of the polygon in RCVG

Figure 7.1.2: Position of the polygon

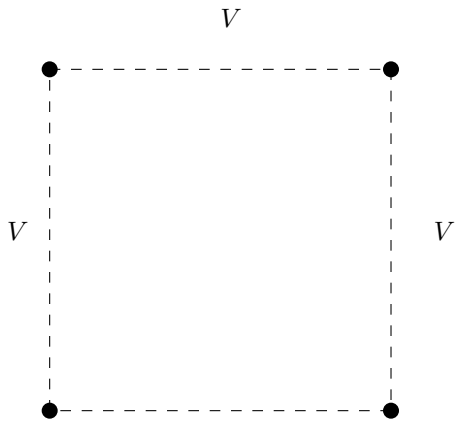


Figure 7.1.3: Vertices of the Polygon

Robots Executing Exact Gathering

The robots executing Exact Gathering form a regular polygon. The distance between all the adjacent vertices is V (Figure 7.1.3). By definition, a robot executing Exact Gathering will not lose visibility with other robots unless the other robot travels more than *limit* distance in one cycle, we exploit this property to construct the polygon. In this construction we can be sure that the robots executing Exact Gathering will gather in the end without losing visibility with each other.

7.2 Impact of Exact Gathering

The results are recorded in Table 7.2.1, the line plot representing data collected from 800 simulations is illustrated in Figure 7.2.1.

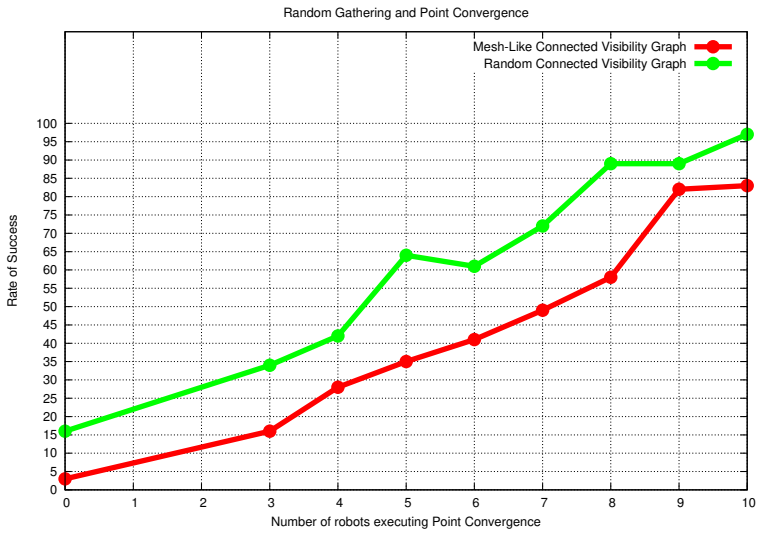


Figure 7.2.1: Results of Random Gathering and Exact Gathering

$\lvert\mathcal{P}\rvert$	Rate of Success
0	3%
3	16%
4	28%
5	35%
6	41%
7	49%
8	58%
9	82%
10	83%

(a) Rate of Success in MCVG

$\lvert\mathcal{P}\rvert$	Rate of Success
0	16%
3	34%
4	42%
5	64%
6	61%
7	72%
8	89%
9	89%
10	97%

(b) Rate of Success in RCVG

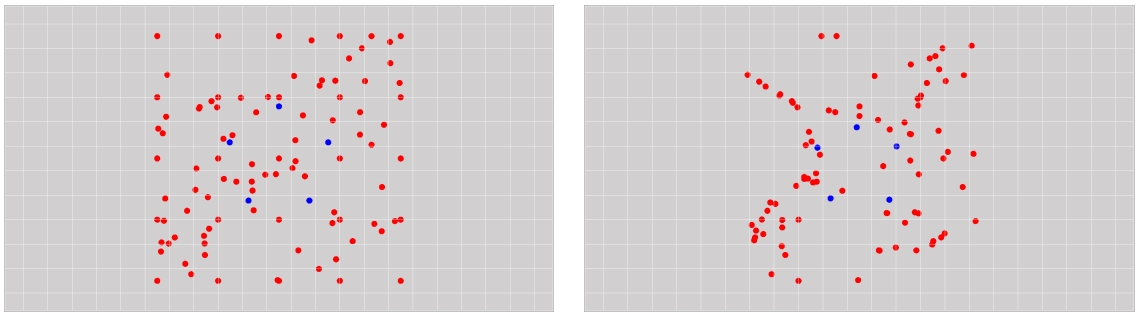
Table 7.2.1: Results of Random Gathering and Exact Gathering

The results we have observed suggests a definite increase in the rate of success compared to the rate of success in Simple Random Gathering algorithm. The smallest polygon with only three vertices is able to increase the rate of gathering by 13% in MCVG

and 18% in RCVG. Increase in the number of robots executing Exact Gathering does not just increase the vertices of the polygon but also the area covered by the polygon since the distance between two adjacent vertices of the polygon is V at t_0. The near linear increase in plot lines suggests that the rate of success increases with the increase in the area covered by the polygon. The rate of success is lower in MCVG because of its initial uniformity. We observe robots in $\mathcal{G}$ and $\mathcal{P}$ exhibit new behaviors which they do not independently, these behaviors are common to MCVG and RCVG;

1. **Polygon and Exact Gathering**

   The polygon initially formed does not collapse immediately to converge at the center as is the case in pure Exact Gathering, instead the polygon is stable for a few cycles with minimal distortions. The polygon does not expand because the vertices cannot lose visibility with each other, at the same time it cannot quickly collapse because the robots may lose visibility with the other robots outside the polygon.



(a) At time t_0 (b) At time t_3

Figure 7.2.2: Behavior of the Polygon

2. **Polygon and Random Gathering**

   Robots executing Random Gathering tend to take positions at vertices of the polygon, since robots on the vertices move slowly the robots executing Random Gathering view them at the same position in multiple cycles giving them a higher chance of being selected as destinations.

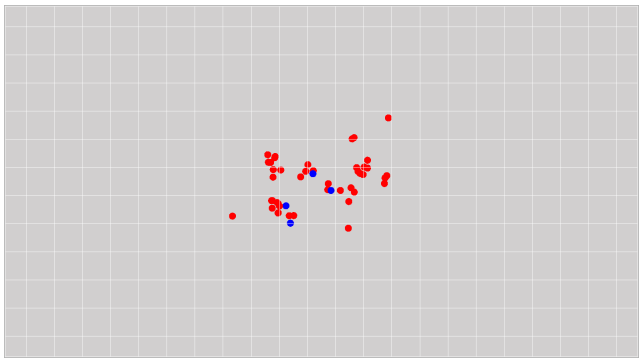


Figure 7.2.3: Robots executing Random Gathering move towards the vertices

3. **Robots clear the center**

   This behavior is counter intuitive and works opposite to what we have observed in Random Gathering. The robots which were initially closer to the center move outward towards the polygon and take positions with the robots executing Exact Gathering at the vertices.

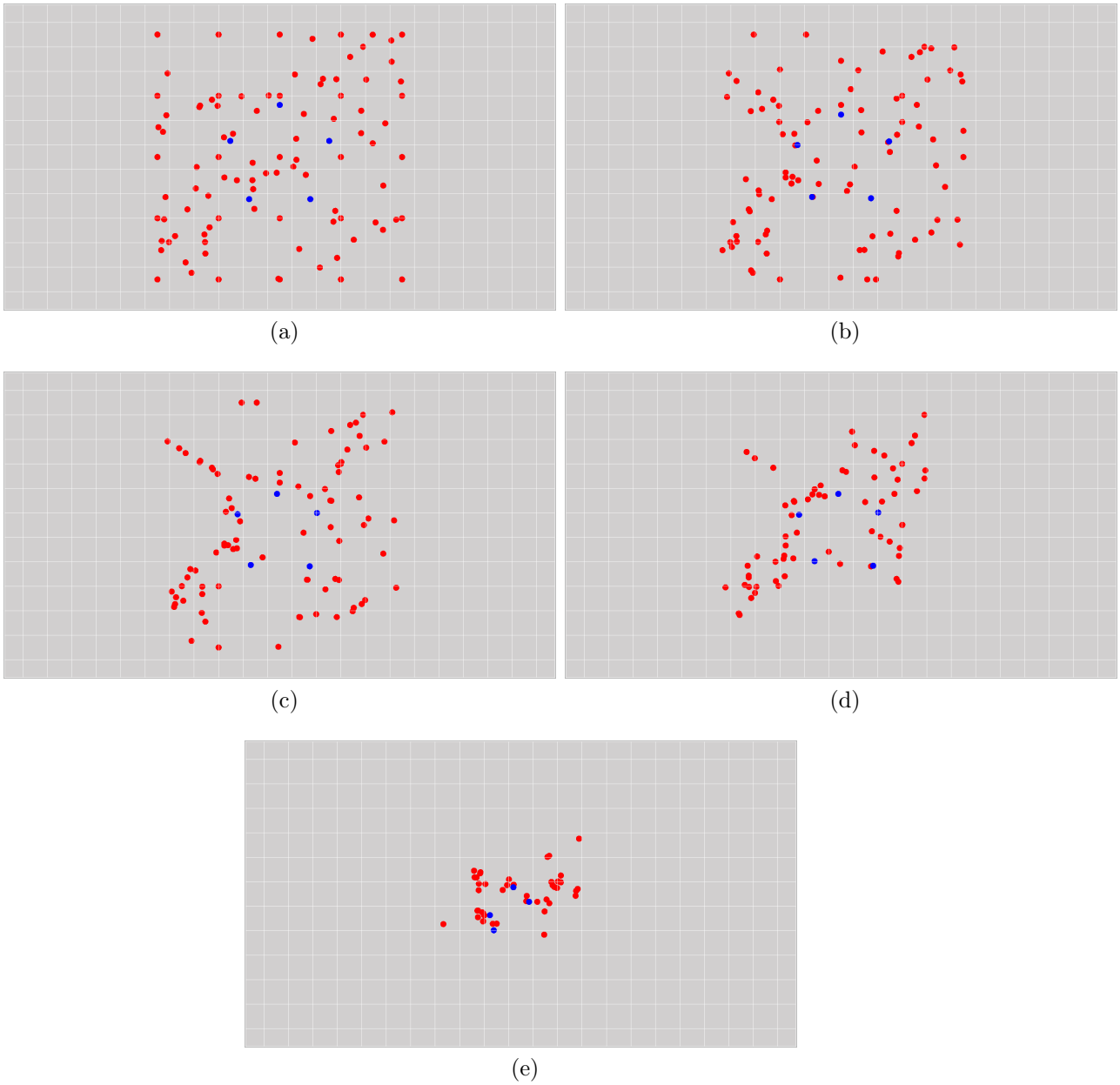
4. **Exact Gathering guides Random Gathering.**

   When all the robots from outside the polygon take positions on vertices of the polygon, the robots executing Exact Gathering move closer to the center, this forces the robots executing Random Gathering to take positions in the vertices of the new polygon and thus constantly shrinks the size of the configuration.

5. **Increasing influence of Exact Gathering**

   The influence of robots executing Exact Gathering on robots executing Random Gathering increases as the size of the polygon decreases, when the robots executing Exact Gathering are closer to each other, the smaller polygon creates a crowded region which is favored by the Random Gathering algorithm. Eventually, the robots executing Exact Gathering gather to a single position creating a crowd at that position which, due to multiplicity detection, is highly favored by Random Gathering. Once the robots executing Exact Gathering gather to a single position

they cannot split thereafter.



(a) (b) (c) (d) (e)

Figure 7.2.4: The Simulation

Figure 7.2.4 is a collection of screen shots captured in intervals during a simulation, it graphically illustrates robot movement and properties discussed above.

7.3 Conclusion

We observed similarity in robot movements in Random Gathering and Exact Gathering, in this Chapter we performed experiments to verify whether the two algorithms could be combined in the same execution. The combination is helpful when many robots with limited resources and only a few with more resources are available, it does not improve the Exact Gathering algorithm but the rate of success is better than Simple Random Gathering. We observe the rate of success improved with as few as three robots executing Exact Gathering algorithm. The experiments show new behaviors in robots which are not observed with either of the two algorithms independently.

It will be interesting to study the rate of success with concentric polygons such that the visibility graph of every robot executing Exact Gathering is connected at time t_0 for $|\mathcal{P}| \geq 6$. It is possible to combine other algorithms, determining a measure of algorithm compatibility would be an interesting study.

Chapter 8

Conclusions and Open Directions

In this thesis we considered Sycamore, a simulator for autonomous mobile robots recently developed to execute and visualize the execution of coordination algorithms. Our goal was to include new features to Sycamore and to start using it to understand the robots' behavior in some novel situations.

After implementing several new features for Sycamore, we focused on Gathering by robots with limited visibility and we performed experiments in three different situations:

1. We implemented a well known deterministic algorithm [3], examining it for the first time in presence of faulty robots; the algorithm has been observed in linear (one-dimensional) settings, as well as in the plane. In linear settings we observed, in particular, the robots' behavior in presence of two faulty robots arbitrary placed, noticing that the healthy robots always form a stable pattern between the two faulty robots. In 2-dimensional settings we experimented with randomly placed robots, but we especially considered faulty robots in regular forms (polygons) noticing that their presence often gave rise to oscillations between patterns by the healthy robots.

2. We implemented and analyzed a new simple random algorithm under different initial configurations. We started by the standard Sycamore "random" conditions for an initial configuration with a connected visibility graph, noticing an unexpected

success rate of the algorithm. We linked the reasons for the success to the non-homogeneity of the initial placement of the robots. We created two new classes of initial conditions and we compared, through many simulations, the success rate of the algorithm in the three different cases.

3. We finally observed the behavior of the robots when combining the two previous algorithms. The idea was to see how robots executing the exact algorithm would influence positively the success rate of the random algorithm. We did run simulations under different conditions and we observed the interaction between the two different sets of rules. The robots executing the exact algorithm act as "guides", leading the other robots towards the interior of their convex hull, thus increasing the probability of cluster creations, and thus the rate of success.

4. We significantly contributed to the development of Sycamore in terms of new features (for example, the creation of a batch simulator to allow extensive experimentation), intuitive and reusable plugins, as well as indications for future development.

For each algorithm we implemented, we made new observations opening the door to several theoretical problems worth studying. Particularly interesting facts that should be further explored are the following:

1. The exact gathering algorithm described in [3] and studied in Chapter 5 has an unexpected and interesting behavior when some of the robots are faulty and do not move for the entire execution. If the faulty robots are close enough to each other, the healthy robots perform gathering regardless of their presence. In all other cases, as expected, they fail to gather. What is perhaps unexpected is the observed robots' behavior when they fail to gather. All our experiments have revealed either the formation of a stable pattern inside the convex hull of the faulty robots, or an oscillating behavior between two patterns. These observations leave open the study of the robots' dynamics: why do we observe only oscillations of period 2? Given

a set of faulty robots in arbitrary positions, what is the set of stable patterns? In general, the systematic study of the robots behavior is left open and this seems a very interesting area of investigation.

2. The observations of the execution of the simple random gathering algorithm of Chapter 6 allowed us to see a clear link between success rate and initial conditions of the robots. Initial setting containing a particular cluster of robots are naturally creating better conditions for the success of the algorithm, while more homogeneous initial configurations are prone to make the algorithm fail. The natural open problem would be to formalize the three different initial configuration settings (only informally described in Chapter 6) and to analyze the algorithm theoretically to see what is its probability of success under the three scenarios.

3. The combination of the two algorithms has been done mainly to test Sycamore's ability to implement hybrid solutions. This idea could be exploited further. Are there situations where it would be advantageous to have groups of heterogeneous robots executing different algorithms towards a common goal? In our situation, a set of robots follow the more complex deterministic algorithm, while the other set follows very simple instructions like flipping a coin. The combination significantly improves the probabilistic solution even with the presence of very few robots following the deterministic algorithm. It would be interesting to explore more the possibility of designing algorithms with heterogeneous robots for other problems.

Bibliography

[1] Chrysovalandis Agathangelou, Chryssis Georgiou, and Marios Mavronicolas. A distributed algorithm for gathering many fat mobile robots in the plane. In *Proceedings of the 32nd ACM Symposium on Principles of Distributed Computing (PODC)*, 250–259, 2013.

[2] Noa Agmon and David Peleg. Fault-Tolerant gathering algorithms for autonomous mobile robots. *SIAM Journal on Computing*, 36(1):56–82, 2006.

[3] Hideki Ando, Yoshinobu Oasa, Ichiro Suzuki, and Masafumi Yamashita. Distributed memoryless point convergence algorithm for mobile robots with limited visibility. *IEEE Transactions on Robotics and Automation*, 15(5):818–828, 1999.

[4] Chanderjit Bajaj. The algebraic degree of geometric optimization problems. *Discrete & Computational Geometry*, 3(1):177–191, 1988.

[5] Eduardo Mesa Barrameda, Shantanu Das, and Nicola Santoro. Deployment of asynchronous robotic sensors in unknown orthogonal environments. In *Proceedings of the 4th International Workshop on Algorithmic Aspects of Wireless Sensor Networks (ALGOSENSOR)*, 125–140, 2008.

[6] Zohir Bouzid, Shantanu Das, and Sébastien Tixeuil. Gathering of Mobile Robots Tolerating Multiple Crash Faults. In *Proceedings of the 33rd International Conference on Distributed Computing Systems (ICDCS)*, 337–346, 2013.

[7] Zohir Bouzid, Maria Potop-Butucaru, and Sébastien Tixeuil. Byzantine convergence in robots networks: The price of asynchrony. In *Proceedings of the 13th International Conference on Principles of Distributed Systems (OPODIS)*, 54–70, 2009.

[8] Zohir Bouzid, Maria Gradinariu Potop-Butucaru, and Sébastien Tixeuil. Optimal byzantine-resilient convergence in uni-dimensional robot networks. *Theoretical Computer Science*, 411(34-36):3154–3168, 2010.

[9] Mark Cieliebak, Paola Flocchini, Giuseppe Prencipe, and Nicola Santoro. Distributed computing by mobile robots: Gathering. *SIAM Journal on Computing*, 41(4):829–879, 2012.

[10] Mark Cieliebak and Giuseppe Prencipe. Gathering autonomous mobile robots. In *Proceedings of the 9th International Colloquium on Structural Information and Communication Complexity (SIROCCO)*, 57–72, 2002.

[11] Reuven Cohen and David Peleg. Convergence properties of the gravitational algorithm in asynchronous robot systems. *SIAM Journal on Computing*, 34(6):1516–1528, 2005.

[12] Reuven Cohen and David Peleg. Convergence of autonomous mobile robots with inaccurate sensors and movements. In *Proceedings of the 23rd International Symposium on Theoretical Aspects of Computer Science (STACS)*, 549–560, 2006.

[13] Reuven Cohen and David Peleg. Local spreading algorithms for autonomous robot systems. *Theoretical Computer Science*, 399(1-2):71–82, 2008.

[14] Andreas Cord-Landwehr, Bastian Degener, Matthias Fischer, Martina Hüllmann, Barbara Kempkes, Alexander Klaas, Peter Kling, Sven Kurras, Marcus Märtens, Friedhelm Meyer auf der Heide, and Others. A new approach for analyzing convergence algorithms for mobile robots. In *Proceedings of the 38th International Colloquium on Automata, Languages and Programming (ICALP)*, 650–661, 2011.

[15] Jurek Czyzowicz, Leszek Gasieniec, and Andrzej Pelc. Gathering few fat mobile robots in the plane. *Theoretical Computer Science*, 410(6-7):481–499, 2009.

[16] Shantanu Das, Paola Flocchini, Giuseppe Prencipe, Nicola Santoro, and Masafumi Yamashita. Autonomous mobile robots with lights. *Theoretical Computer Science*, 609:171–184, 2016.

[17] Shantanu Das, Paola Flocchini, Nicola Santoro, and Masafumi Yamashita. On the computational power of oblivious robots. In *Proceedings of the 29th ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing (PODC)*, 267–276, 2010.

[18] Xavier Défago and Akihiko Konagaya. Circle formation for oblivious anonymous mobile robots with no common sense of orientation. In *Proceedings of the 2nd ACM International workshop on Principles of Mobile Computing (POMC)*, 97–104, 2002.

[19] Bastian Degener, Barbara Kempkes, Tobias Langner, Friedhelm Meyer auf der Heide, Peter Pietrzyk, and Roger Wattenhofer. A tight runtime bound for synchronous gathering of autonomous robots with limited visibility. In *Proceedings of the 23rd ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, 139–148, 2011.

[20] Yoann Dieudonné, Ouiddad Labbani-Igbida, and Franck Petit. Circle formation of weak mobile robots. *ACM Transactions on Autonomous and Adaptive Systems*, 3(4):1–20, 2008.

[21] Yoann Dieudonné and Franck Petit. Squaring the circle with weak mobile robots. In *Proceedings of the 19th International Symposium on Algorithms and Computation (ISAAC)*, 354–365, 2008.

[22] Yoann Dieudonné and Franck Petit. Scatter of weak robots. *Parallel Processing Letters*, 19(1):175–184, 2009.

[23] Yoann Dieudonné and Franck Petit. Self-stabilizing gathering with strong multiplicity detection. *Theoretical Computer Science*, 428:47–57, 2012.

[24] Stephane Durocher and David Kirkpatrick. The projection median of a set of points. *Computational Geometry*, 42(5):364–375, 2009.

[25] Paola Flocchini. Computations by luminous robots. In *Proceedings of the 14th International Conference on Ad-hoc, Mobile, and Wireless Networks (ADHOC-NOW)*, 238–252, 2015.

[26] Paola Flocchini. Memoryless gathering of mobile robotic sensors. In *Encyclopedia of Algorithms*. Springer, 2015.

[27] Paola Flocchini. Uniform covering of rings and lines by memoryless mobile sensors. In *Encyclopedia of Algorithms*. Springer, 2015.

[28] Paola Flocchini, Giuseppe Prencipe, and Nicola Santoro. Self-deployment of mobile sensors on a ring. *Theoretical Computer Science*, 402(1):67–80, 2008.

[29] Paola Flocchini, Giuseppe Prencipe, and Nicola Santoro. *Distributed Computing by Oblivious Mobile Robots*. Synthesis lectures on distributed computing theory. Morgan & Claypool, 2012.

[30] Paola Flocchini, Giuseppe Prencipe, Nicola Santoro, and Giovanni Viglietta. Distributed computing by mobile robots: Solving the uniform circle formation problem. In *Proceedings of the 18th International Conference on Principles of Distributed Systems (OPODIS)*, 217–232, 2014.

[31] Paola Flocchini, Giuseppe Prencipe, Nicola Santoro, and Peter Widmayer. Gathering of asynchronous robots with limited visibility. *Theoretical Computer Science*, 337(1-3):147–168, 2005.

[32] Paola Flocchini, Giuseppe Prencipe, Nicola Santoro, and Peter Widmayer. Arbitrary pattern formation by asynchronous, anonymous, oblivious robots. *Theoretical Computer Science*, 407(1-3):412–447, 2008.

[33] Paola Flocchini, Nicola Santoro, Giovanni Viglietta, and Masafumi Yamashita. Rendezvous with constant memory. *Theoretical Computer Science*, In press.

[34] Nao Fujinaga, Hirotaka Ono, Shuji Kijima, and Masafumi Yamashita. Pattern formation through optimum matching by oblivious CORDA robots. In *Proceedings of the 14th International Conference on Principles of Distributed Systems (OPODIS)*, 1–15, 2010.

[35] Sruti Gan Chaudhuri and Krishnendu Mukhopadhyaya. Leader election and gathering for asynchronous fat robots without common chirality. *Journal of Discrete Algorithms*, 33:171–192, 2015.

[36] A. Ganguli, J. Cortes, and F. Bullo. Multirobot rendezvous with visibility sensors in nonconvex environments. *IEEE Transactions on Robotics*, 25(2):340–352, 2009.

[37] Taisuke Izumi, Zohir Bouzid, Sébastien Tixeuil, and Koichi Wada. Brief Announcement: The BG-simulation for byzantine mobile robots. In *Proceedings of the 25th International Symposium on Distributed Computing (DISC)*, 330–331, 2011.

[38] Taisuke Izumi, Yoshiaki Katayama, Nobuhiro Inuzuka, and Koichi Wada. Gathering autonomous mobile robots with dynamic compasses: An optimal result. In *Proceedings of the 21st International Symposium on Distributed Computing (DISC)*, 298–312, 2007.

[39] Taisuke Izumi, Maria Gradinariu Potop-Butucaru, and Sébastien Tixeuil. Connectivity-preserving scattering of mobile robots with limited visibility. In *Proceedings of the 12th International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS)*, 319–331, 2010.

[40] Taisuke Izumi, Samia Souissi, Yoshiaki Katayama, Nobuhiro Inuzuka, Xavier Défago, Koichi Wada, and Masafumi Yamashita. The gathering problem for two oblivious robots with unreliable compasses. *SIAM Journal on Computing*, 41(1):26–46, 2012.

[41] Branislav Katreniak. Convergence with limited visibility by asynchronous mobile robots. In *Proceedings of the 18th International Colloquium on Structural Information and Communication Complexity (SIROCCO)*, 125–137, 2011.

[42] Giuseppe Antonio Di Luna, Paola Flocchini, Sruti Gan Chaudhuri, Federico Poloni, Nicola Santoro, and Giovanni Viglietta. Mutual visibility by luminous robots without collisions. *Information and Computation*, In press.

[43] Sonia Martínez. Practical multiagent rendezvous through modified circumcenter algorithms. *Automatica*, 45(9):2010–2017, 2009.

[44] A Stephen Morse, Brian Anderson, and Jie Lin. The multi-agent rendezvous problem. Part 1: The synchronous case. *SIAM Journal on Control and Optimization*, 46(6):2096–2119, 2007.

[45] A Stephen Morse, Brian Anderson, and Jie Lin. The multi-agent rendezvous problem. Part 2: The asynchronous case. *SIAM Journal on Control and Optimization*, 46(6):2120–2147, 2007.

[46] Linda Pagli, Giuseppe Prencipe, and Giovanni Viglietta. Getting close without touching. In *Proceedings of the 19th International Colloquium on Structural Information and Communication Complexity (SIROCCO)*, 315–326, 2012.

[47] Giuseppe Prencipe. Impossibility of gathering by a set of autonomous mobile robots. *Theoretical Computer Science*, 384(2-3):222–231, 2007.

[48] Samia Souissi. *Fault-resilient cooperation of autonomous mobile robots with unreliable compass sensors.* PhD thesis, Japan Advanced Institute of Science and Technology, 2007.

[49] Samia Souissi, Xavier Défago, and Masafumi Yamashita. Using eventually consistent compasses to gather memory-less mobile robots with limited visibility. *ACM Transactions on Autonomous and Adaptive Systems*, 4(1):1–27, 2009.

[50] Ichiro Suzuki and Masafumi Yamashita. Distributed anonymous mobile robots: formation of geometric patterns. *SIAM Journal on Computing*, 28(4):1347–1363, 1999.

[51] Giovanni Viglietta. Rendezvous of two robots with visible bits. In *Proceedings of the 9th International Symposium on Algorithms and Experiments for Sensor Systems, Wireless Networks and Distributed Robotics (ALGOSENSORS)*, 291–306, 2013.

[52] Valerio Volpi and Giuseppe Prencipe. 2D/3D Mobile robots simulation environment. Masters Thesis. University of Pisa, 2013.

[53] Endre Weiszfeld. Sur le point pour lequel la somme des distances de n points donnés est minimum. *Tohoku Math*, 43(355-386):2, 1937.