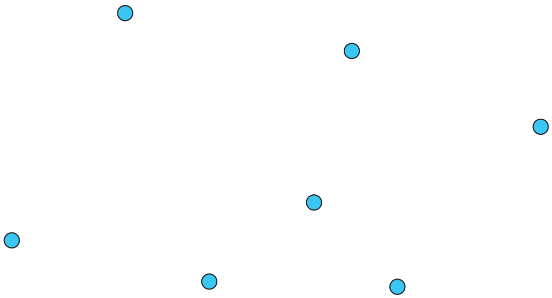
Introduction to Mobile Robots

Abstract Model and Classical Problems

Giovanni Viglietta

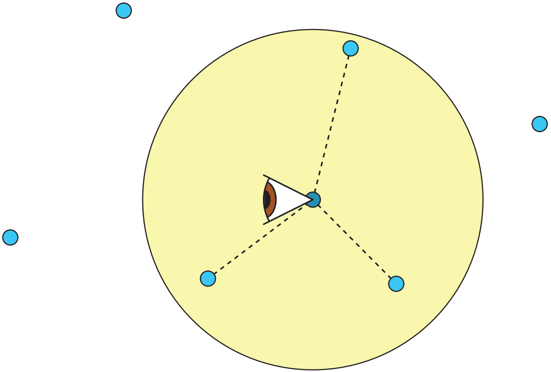
JAIST – November 22, 2018

Anonymous robots sensing and moving



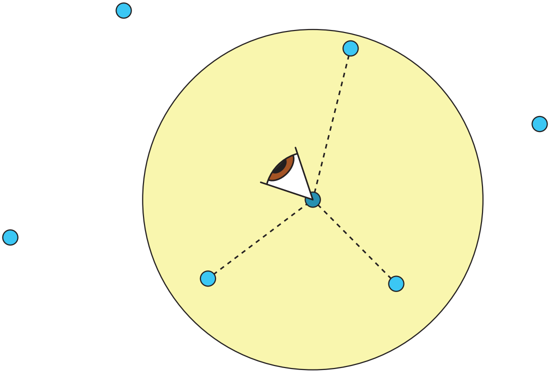
We consider a *swarm* of anonymous robots in a Euclidean space.

Anonymous robots sensing and moving



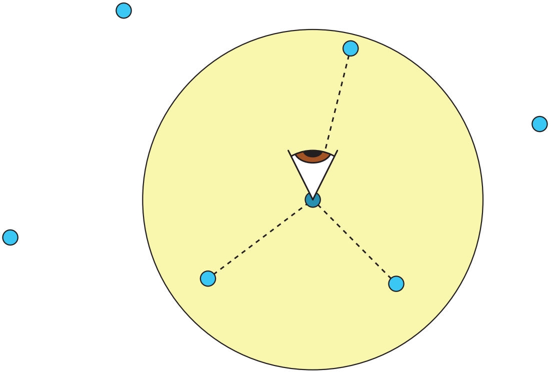
Each robot can see the positions of all robots within a range...

Anonymous robots sensing and moving



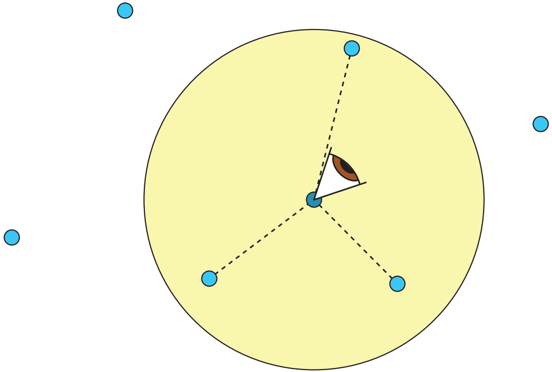
Each robot can see the positions of all robots within a range...

Anonymous robots sensing and moving



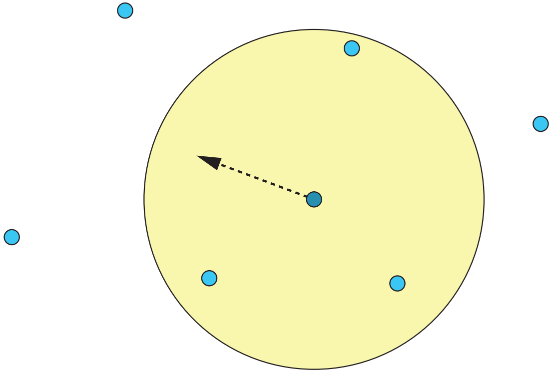
Each robot can see the positions of all robots within a range...

Anonymous robots sensing and moving



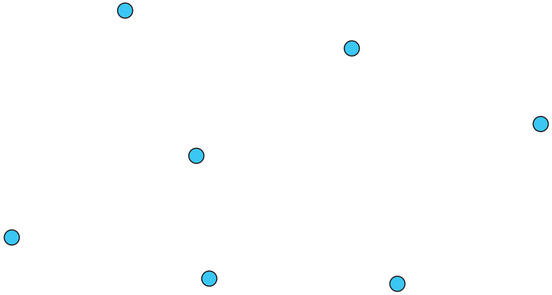
Each robot can see the positions of all robots within a range...

Anonymous robots sensing and moving



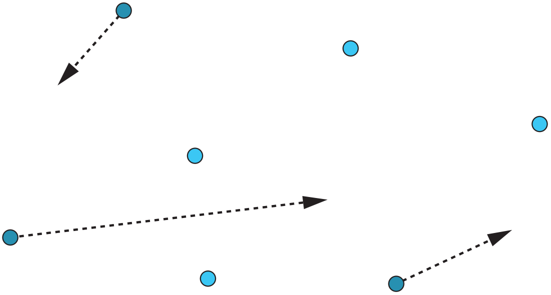
...And move according to a <u>deterministic</u> algorithm.

Anonymous robots sensing and moving



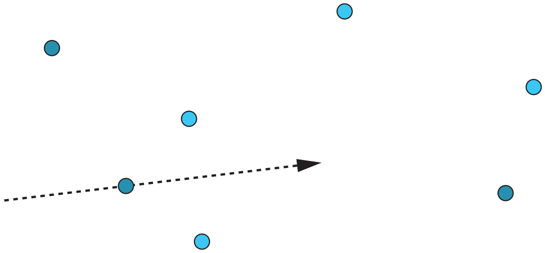
...And move according to a <u>deterministic</u> algorithm.

Anonymous robots sensing and moving



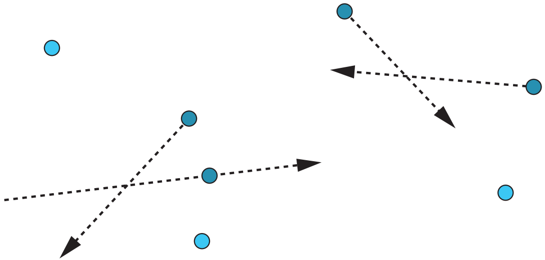
Different robots are activated asynchronously.

Anonymous robots sensing and moving



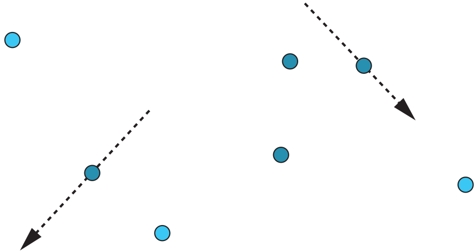
Different robots are activated asynchronously.

Anonymous robots sensing and moving



Different robots are activated asynchronously.

Anonymous robots sensing and moving



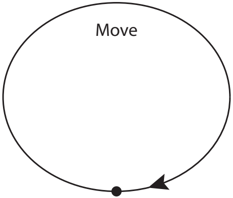
Different robots are activated asynchronously.

Model definition

Robots are:

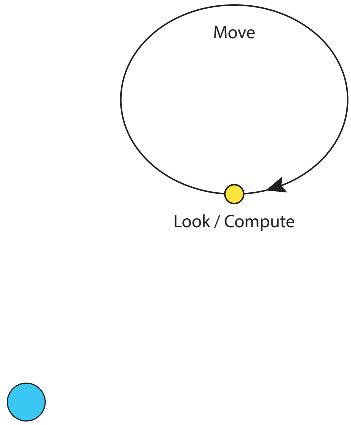
- **Dimensionless** (robots are modeled as geometric points)
- **Anonymous** (no unique identifiers)
- **Homogeneous** (the same algorithm is executed by all robots)
- **Autonomous** (no centralized control)
- **Oblivious** (no memory of past events and observations)
- **Silent** (no explicit way of communicating)
- **Short-sighted** (visibility of other robots limited to a range)
- **Disoriented** (robots do not share a common reference frame)

Life cycle and asynchrony



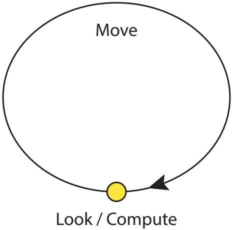
Each robot perpetually repeats a Look/Compute/Move cycle.

Life cycle and asynchrony



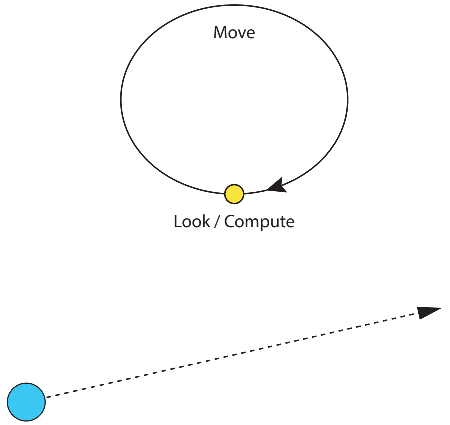
Each robot perpetually repeats a Look/Compute/Move cycle.

Life cycle and asynchrony



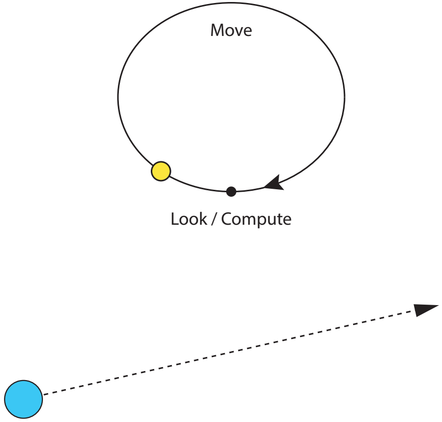
In a Look phase, a snapshot is taken of all visible robots.

Life cycle and asynchrony



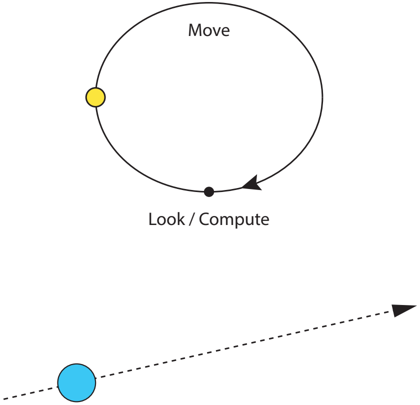
A destination is computed based only on the last snapshot.

Life cycle and asynchrony



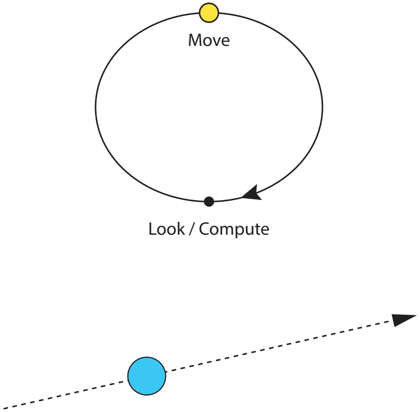
The destination point is approached with unpredictable speed.

Life cycle and asynchrony



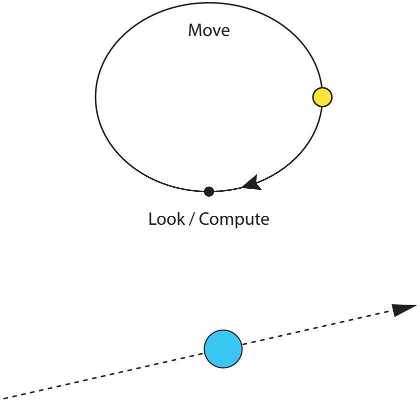
The destination point is approached with unpredictable speed.

Life cycle and asynchrony



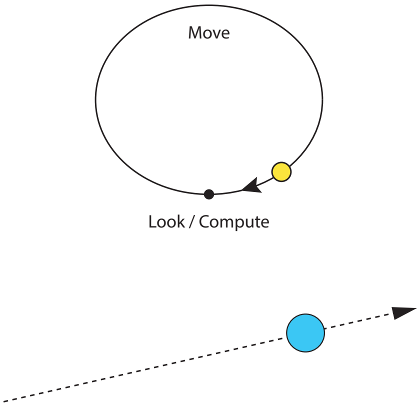
The destination point is approached with unpredictable speed.

Life cycle and asynchrony



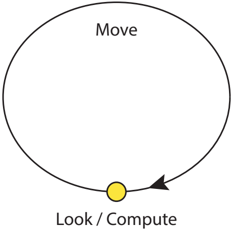
The destination point is approached with <u>unpredictable speed</u>.

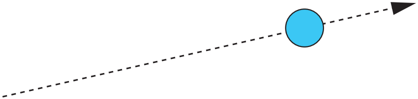
Life cycle and asynchrony



The destination point is approached with unpredictable speed.

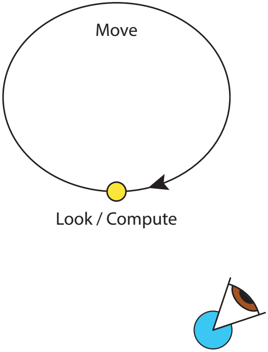
Life cycle and asynchrony





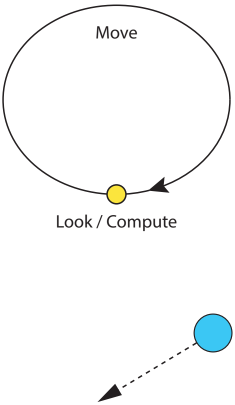
The robot may unpredictably stop before reaching the destination...

Life cycle and asynchrony



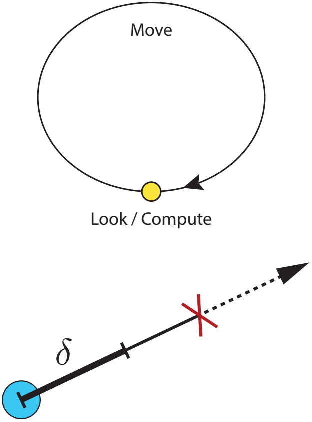
...and execute a new Look/Compute phase.

Life cycle and asynchrony



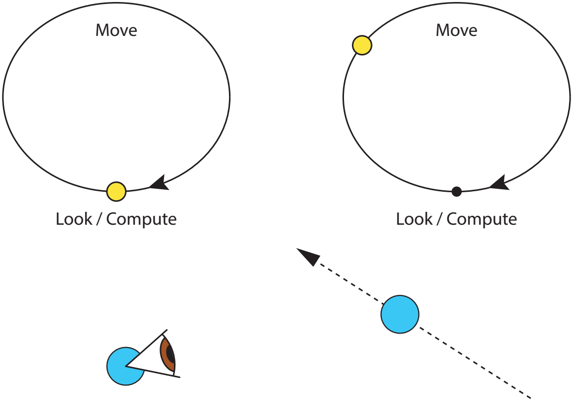
...and execute a new Look/Compute phase.

Life cycle and asynchrony



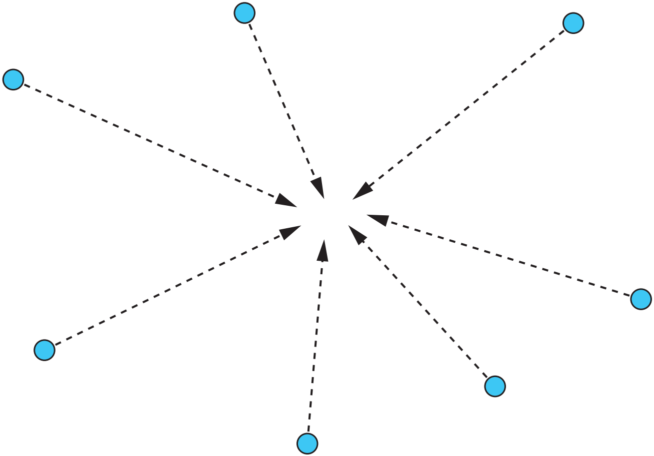
At each cycle, a robot is guaranteed to move by at least δ.

Life cycle and asynchrony



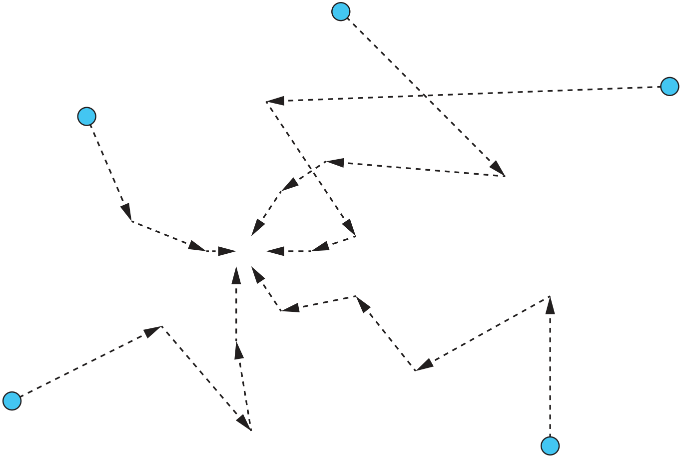
Different robots execute independent cycles, asynchronously.

Gathering-like problems



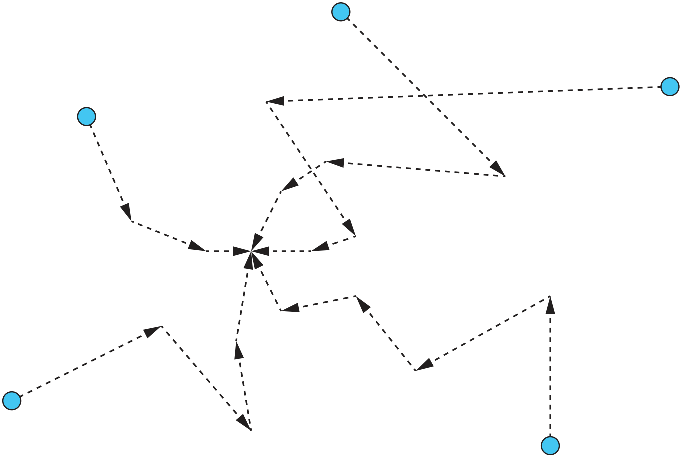
Perhaps the most studied class of problems:
Design an algorithm that makes all robots "gather".

Convergence problem

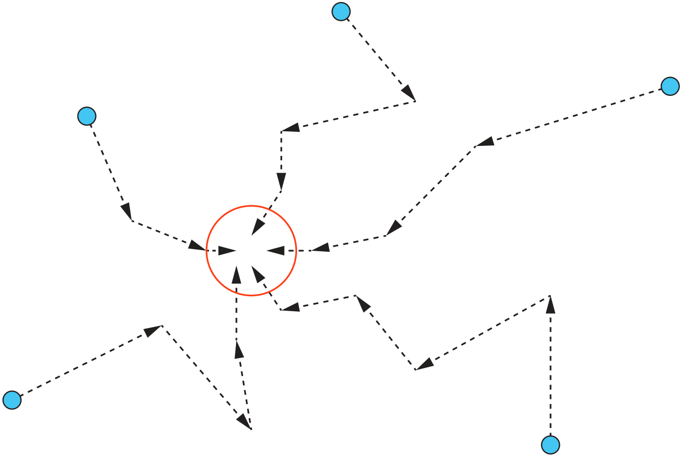


Problem: Make all robots converge to the same point (possibly never reaching it, and possibly colliding).

Gathering problem



Problem: Make all robots reach the same point in finite time.

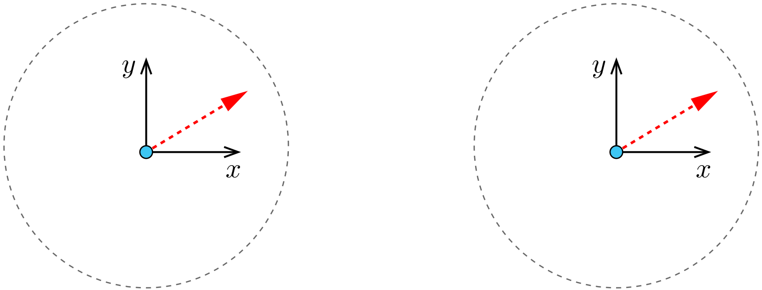


Problem: Make all robots reach a small-enough area, avoiding collisions.

Case study: two robots

Suppose the swarm consists of only two robots.

Case study: two robots



Observation: We must assume that they see each other initially.
If they do not, they may never meet.

Case study: two robots

Strategy: Move to the midpoint.

Case study: two robots

It solves the Gathering problem if robots are fully synchronous...

Case study: two robots

...But it only solves Convergence if robots are asynchronous.

Case study: two robots

…But it only solves Convergence if robots are asynchronous.

Case study: two robots

...But it only solves Convergence if robots are asynchronous.

...But it only solves Convergence if robots are asynchronous.

Case study: two robots

…But it only solves Convergence if robots are asynchronous.

Case study: two robots

...But it only solves Convergence if robots are asynchronous.

Case study: two robots

...But it only solves Convergence if robots are asynchronous.

Case study: two robots

...But it only solves Convergence if robots are asynchronous.

Case study: two robots

Modified strategy: Reduce the distance by a fraction of $1/3$.

Case study: two robots

It solves Near-Gathering if robots are fully synchronous...

Case study: two robots

It solves Near-Gathering if robots are fully synchronous...

Case study: two robots

It solves Near-Gathering if robots are fully synchronous...

Case study: two robots

…But the robots may collide if they are asynchronous.

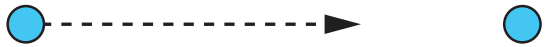
Case study: two robots

...But the robots may collide if they are asynchronous.

Case study: two robots

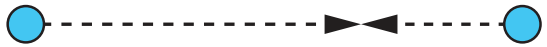
…But the robots may collide if they are asynchronous.

Case study: two robots



...But the robots may collide if they are asynchronous.

Case study: two robots



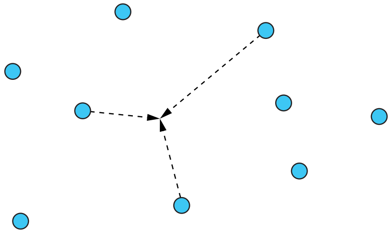
...But the robots may collide if they are asynchronous.

…But the robots may collide if they are asynchronous.

R. Cohen, D. Peleg
Convergence Properties of the Gravitational Algorithms in Asynchronous Robot Systems
SIAM Journal on Computing 34(6):1516–1528, 2005

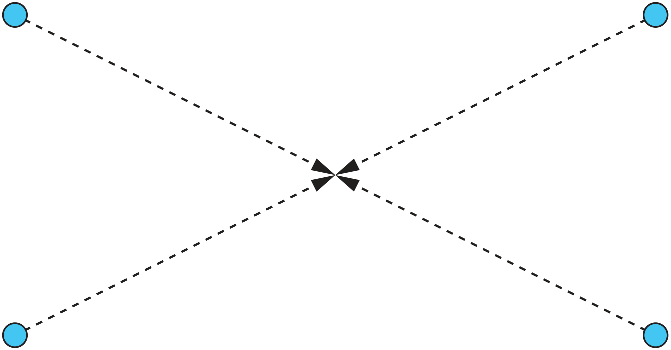
Strategy: Move to the center of gravity.

Requirement: The visibility range must be infinite.

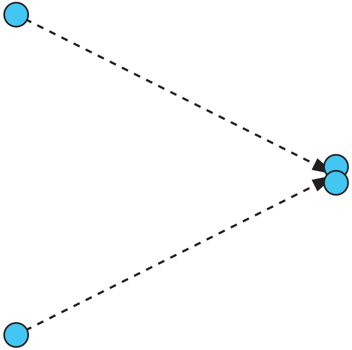


Simple Convergence algorithm: example

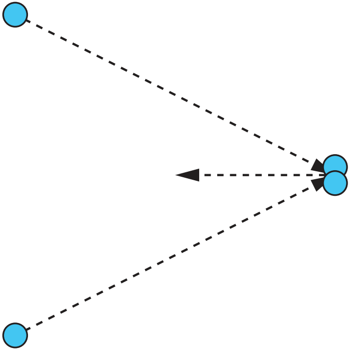
Simple Convergence algorithm: example



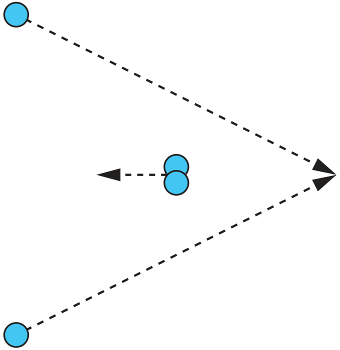
Simple Convergence algorithm: example



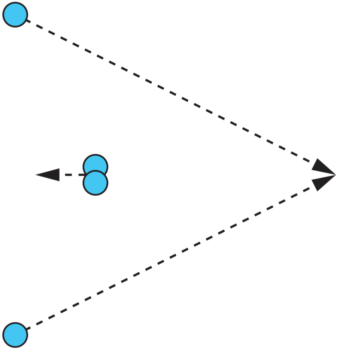
Simple Convergence algorithm: example



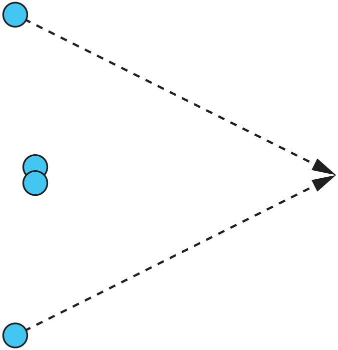
Simple Convergence algorithm: example



Simple Convergence algorithm: example



Simple Convergence algorithm: example



Simple Convergence algorithm: example

Simple Convergence algorithm: example

Simple Convergence algorithm: example

Convergence with limited visibility

H. Ando, Y. Oasa, I. Suzuki, M. Yamashita

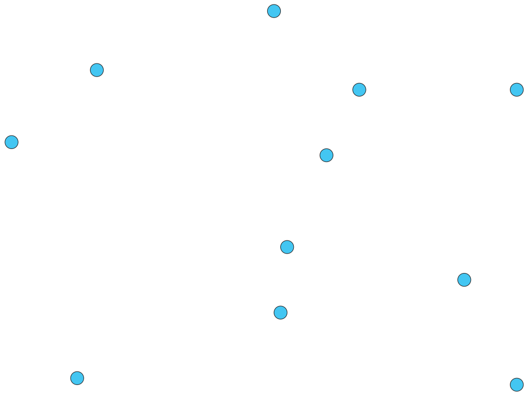
Distributed Memoryless Point Convergence Algorithm for Mobile Robots with Limited Visibility

IEEE Trans. Robot. Autom. 15(5):818–828, 1999

Strategy: Move to the center of the smallest enclosing circle of the visible robots, without going too far from any of them.

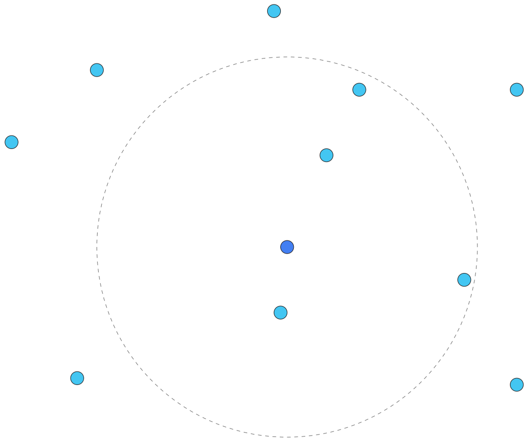
Requirement: Each move must be "instantaneous" (i.e., no robot can see another robot in the middle of a movement).

Convergence with limited visibility: details



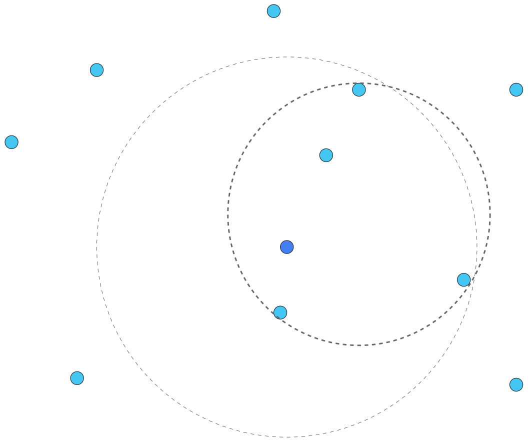
Each robot moves in the direction of the center of the smallest enclosing circle of the visible robots.

Convergence with limited visibility: details



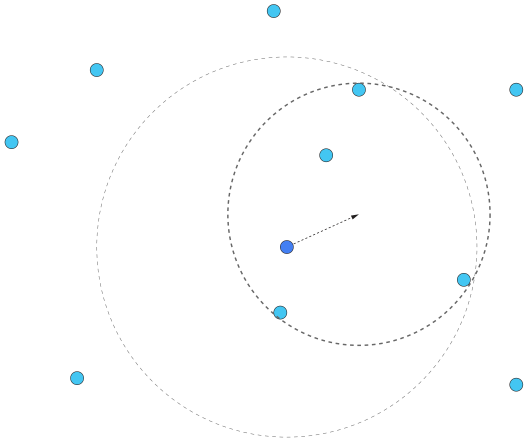
Each robot moves in the direction of the center of the smallest enclosing circle of the visible robots.

Convergence with limited visibility: details



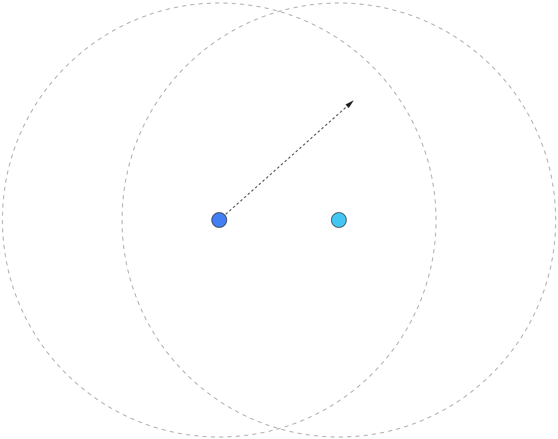
Each robot moves in the direction of the center of the smallest enclosing circle of the visible robots.

Convergence with limited visibility: details



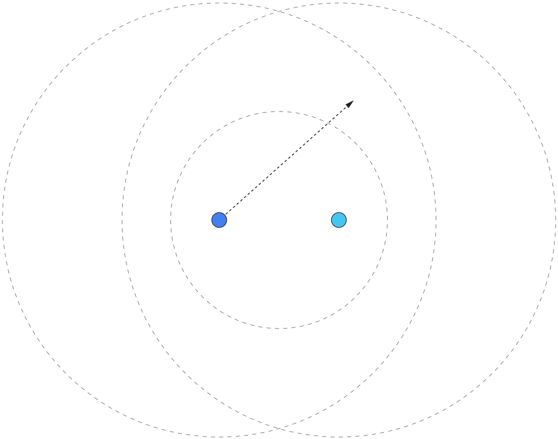
Each robot moves in the direction of the center of the smallest enclosing circle of the visible robots.

Convergence with limited visibility: details



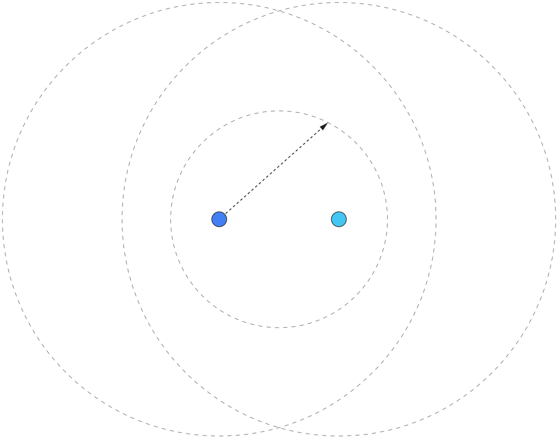
However, it does not get too far from other visible robots.

Convergence with limited visibility: details



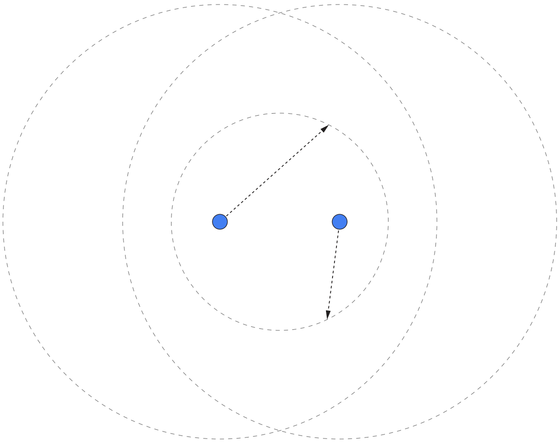
For each other visible robot, it remains within the circle centered at their midpoint and with diameter equal to their radius of visibility.

Convergence with limited visibility: details



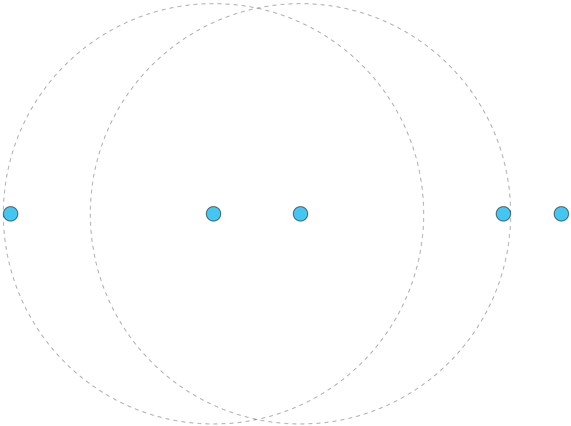
For each other visible robot, it remains within the circle centered at their midpoint and with diameter equal to their radius of visibility.

Convergence with limited visibility: details



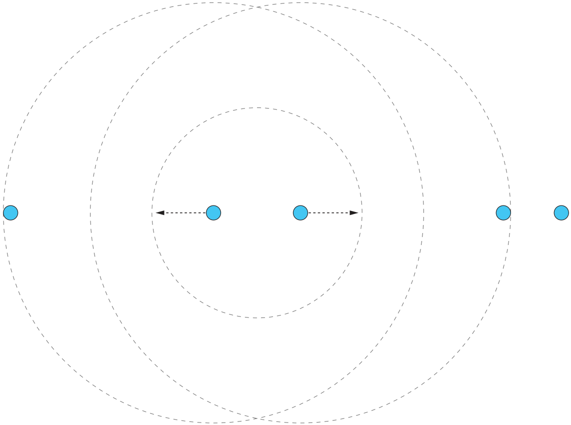
This way, if two robots see each other,
they will keep seeing each other after they move.

The algorithm may fails moves are not instantaneous



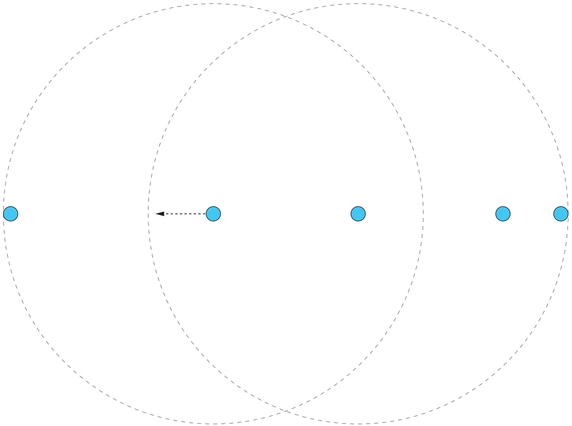
If moves are not instantaneous, some robots may lose vision of each other and converge to different points.

The algorithm may fails moves are not instantaneous



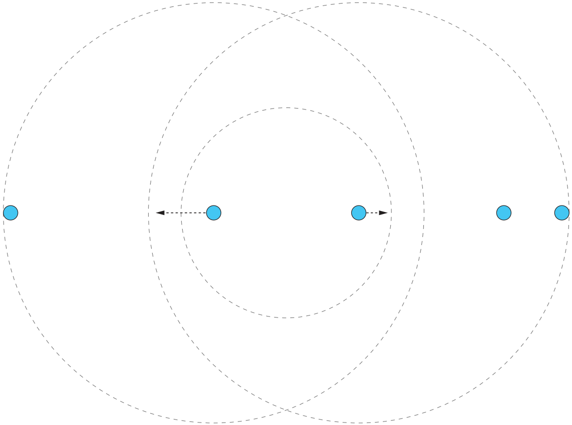
If moves are not instantaneous, some robots may lose vision of each other and converge to different points.

The algorithm may fails moves are not instantaneous



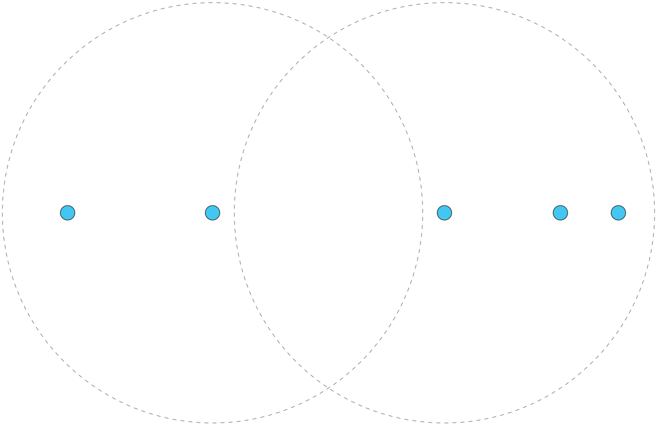
If moves are not instantaneous, some robots may lose vision of each other and converge to different points.

The algorithm may fails moves are not instantaneous



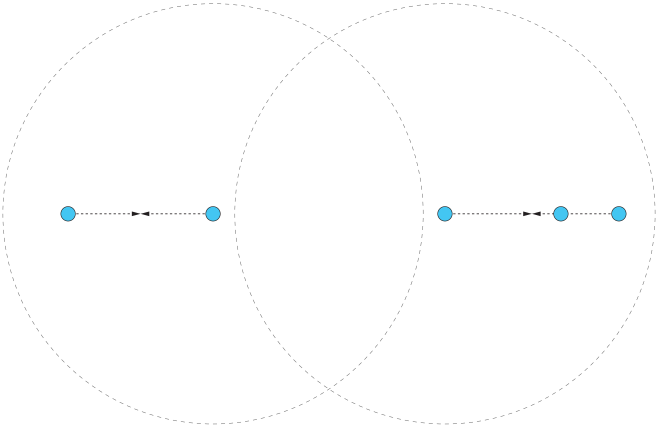
If moves are not instantaneous, some robots may lose vision of each other and converge to different points.

The algorithm may fails moves are not instantaneous



If moves are not instantaneous, some robots may lose vision of each other and converge to different points.

The algorithm may fails moves are not instantaneous



If moves are not instantaneous, some robots may lose vision of each other and converge to different points.

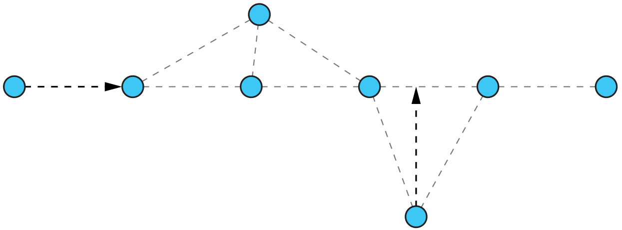
The algorithm may fails moves are not instantaneous

If moves are not instantaneous, some robots may lose vision of each other and converge to different points.

P. Flocchini, G. Prencipe, N. Santoro, P. Widmayer
Gathering of Asynchronous Robots with Limited Visibility
Theoretical Computer Science 337:147–168, 2005

Strategy: If you see only robots "above" you, move upward.
Otherwise, if you see only robots to the "right", move rightward.
Make sure you do not lose vision with other robots.

Requirement: All robots must agree on up and right directions.



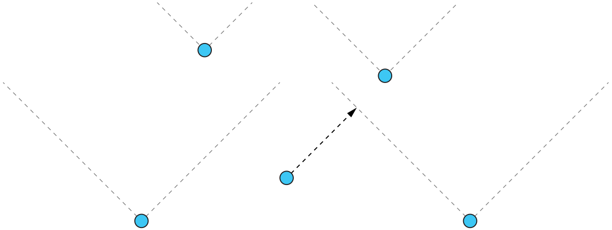
L. Pagli, G. Prencipe, G. Viglietta

Getting Close Without Touching: Near-Gathering for Autonomous Mobile Robots

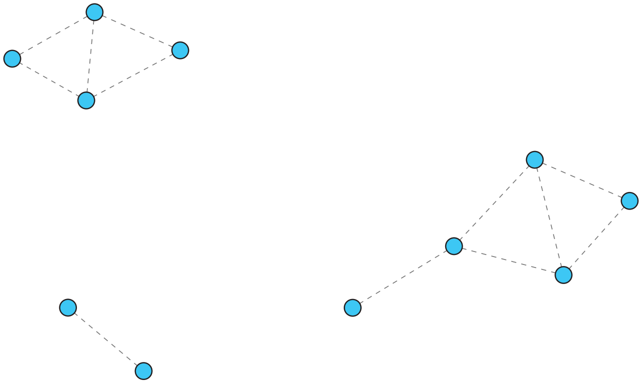
Distributed Computing, 28(5):333–349, 2015

Strategy: Move up-right or up-left without going too far and without entering the "move space" of other robots.

Requirement: All robots must agree on the up direction.



Distance graph



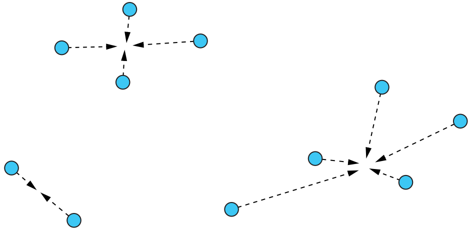
Distance graph: expresses which pairs of robots see each other.

Can the robots gather if their distance graph is not connected?

Assumptions on the distance graph

H. Ando, Y. Oasa, I. Suzuki, M. Yamashita
Distributed Memoryless Point Convergence Algorithm for Mobile Robots with Limited Visibility
IEEE Trans. Robot. Autom. 15(5):818–828, 1999

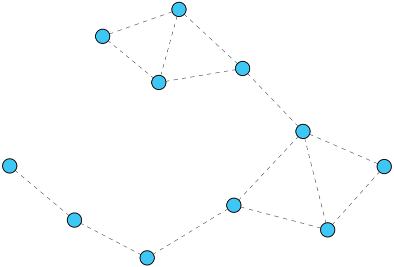
"The objective of a point convergence algorithm is to move the robots in each connected component of the mutual visibility graph to within a sufficiently small neighborhood of a point."



Assumptions on the distance graph

P. Flocchini, G. Prencipe, N. Santoro, P. Widmayer
Gathering of Asynchronous Robots with Limited Visibility
Theoretical Computer Science 337:147–168, 2005

"If the distance graph $D(0)$ is disconnected, the gathering problem is unsolvable. Thus, in the following we will always assume that $D(0)$ is connected."



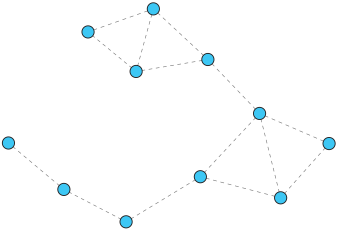
Assumptions on the distance graph

L. Pagli, G. Prencipe, G. Viglietta

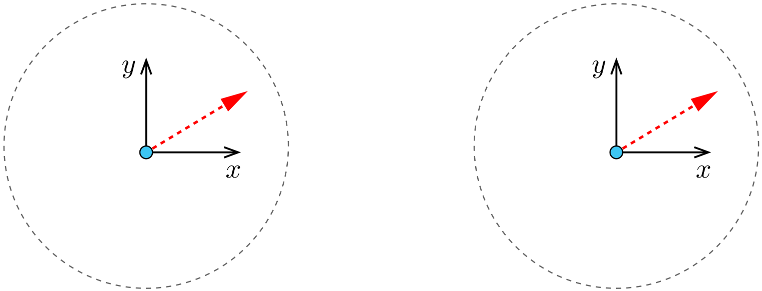
Getting Close Without Touching: Near-Gathering for Autonomous Mobile Robots

Distributed Computing, 28(5):333–349, 2015

"If the initial distance graph I is not connected, the Near-Gathering problem may be unsolvable.
Assumption 1: The initial strong distance graph J is connected."

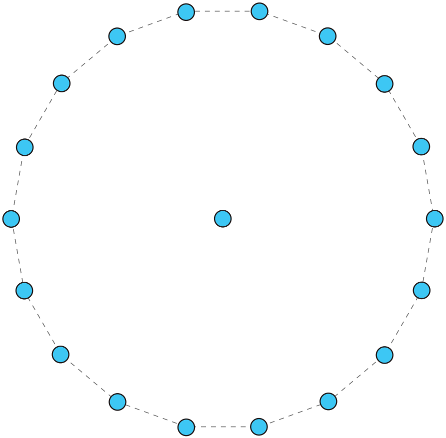


Counterexample



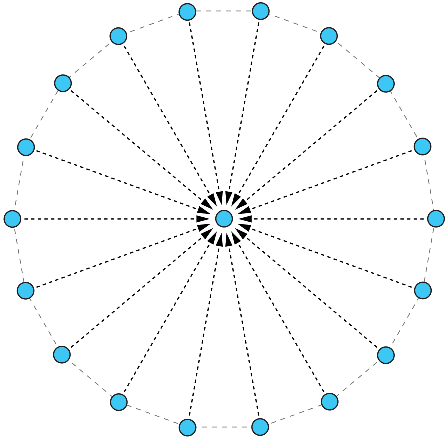
This is the only counterexample given in the literature:
two far-apart robots with the same orientation.
They will keep going in the same direction, and will never meet.

Counter-counterexample



But there are also positive examples.

Counter-counterexample

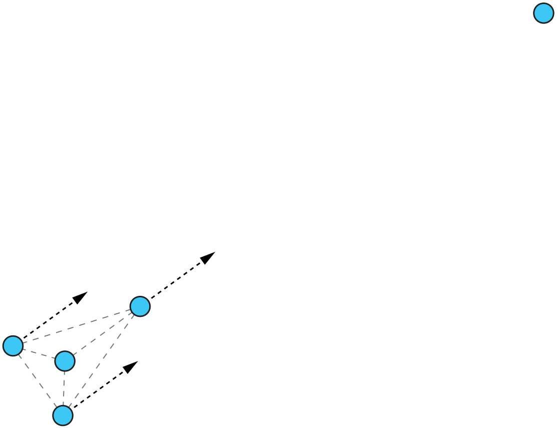


Even existing algorithms will make the robots gather in this case.

Yet another counter-counterexample

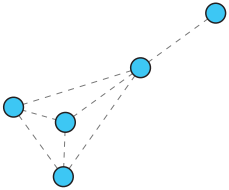
Consider a “module” plus an isolated robot in a strategic location.

Yet another counter-counterexample



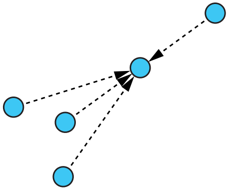
There is an ad-hoc algorithm that makes the module move...

Yet another counter-counterexample



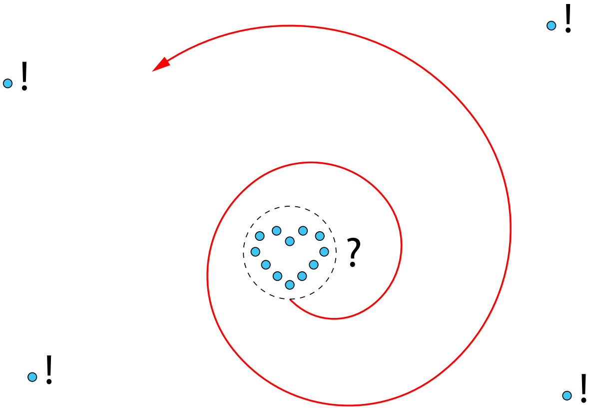
...In such a way as to eventually reach the isolated robot.

Yet another counter-counterexample



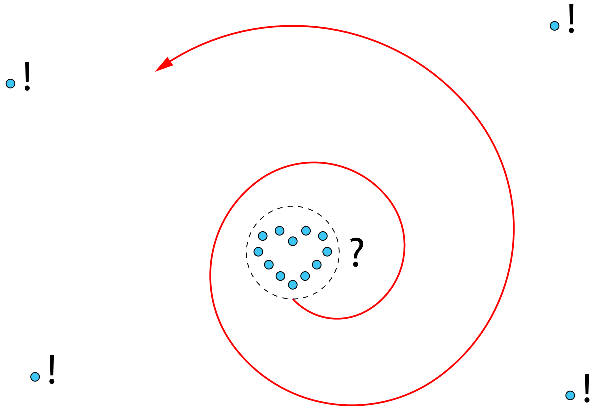
We cannot really say that Gathering is “impossible” in this case!

Programmable module?



Could there be a way to design a "module" that can steer and explore the whole space, collecting all the isolated robots?

Programmable module?



More generally, what computational power can this module have?
Can it be programmable even if the robots are memoryless?

The TuringMobile module

G. A. Di Luna, P. Flocchini, N. Santoro, and G. Viglietta
TuringMobile: A Turing Machine of Oblivious Mobile Robots with Limited Visibility and its Applications
DISC 2018, New Orleans, USA

"Interestingly, the presence of the TuringMobile allows Gathering to be done even if the initial visibility graph is disconnected (this does not change the fact that there are cases in which Gathering is impossible)."

Existence of a TuringMobile

Theorem

If $3\,(m+k)$ *identical robots in* $\mathbb{R}^m$ *with no memory are arranged in a specific pattern and execute a specific algorithm, they can collectively act in the same way as a single robot with* k *registers.*

Moreover, this single robot does not unpredictably stop before reaching its destination point.

Existence of a TuringMobile

Theorem

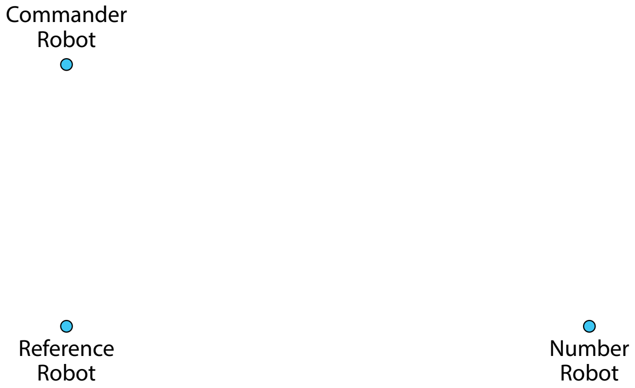
If $3\,(m+k)$ identical robots in $\mathbb{R}^m$ with no memory are arranged in a specific pattern and execute a specific algorithm, they can collectively act in the same way as a single robot with k registers.

Moreover, this single robot does not unpredictably stop before reaching its destination point.

$\Longrightarrow$ A team of unreliable oblivious robots can simulate a single reliable robot with persistent memory.

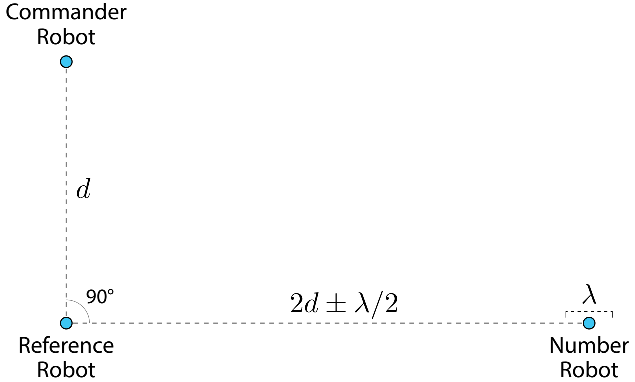
This is effectively a Turing machine that computes and moves through space: the team simulating it is called "TuringMobile".

Basic component of the TuringMobile



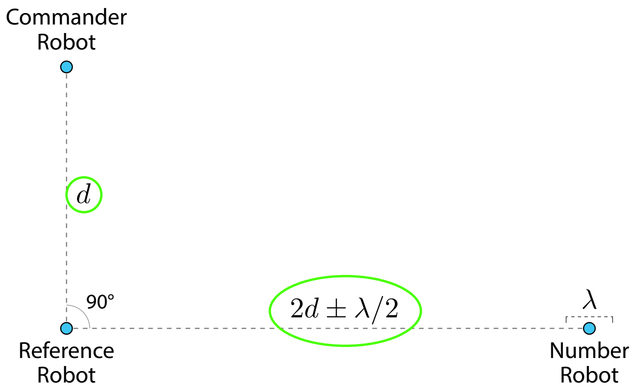
A TuringMobile consists of several copies of a basic component.

Basic component of the TuringMobile



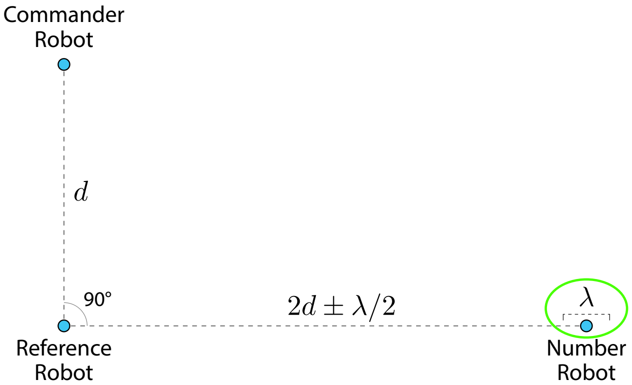
This is the position at rest of the basic component.

Basic component of the TuringMobile



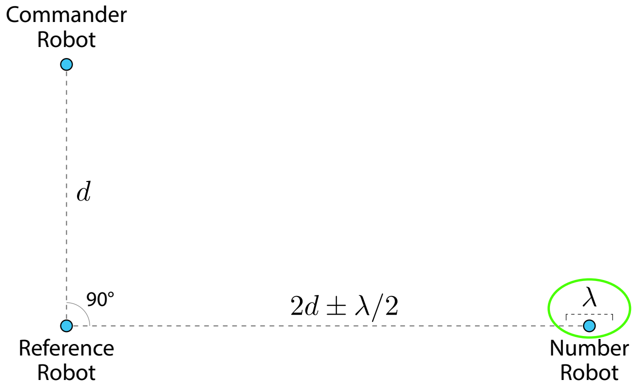
Robots can determine their own identities based on their distances: the Commander and the Reference robot are always closest, etc.

Basic component of the TuringMobile



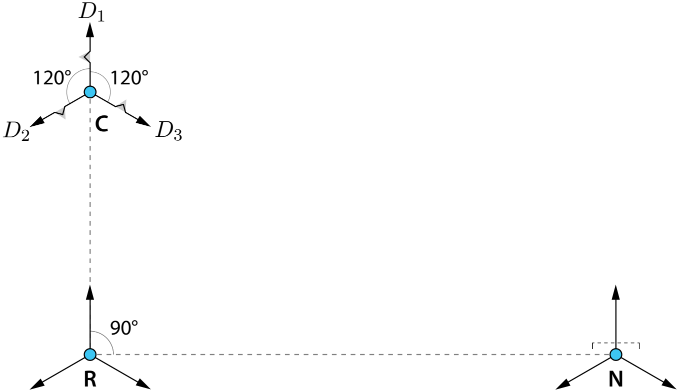
Any <u>real number</u> (i.e., a "state") can be represented by the <u>Number robot</u> based on its position along a small segment.

Basic component of the TuringMobile



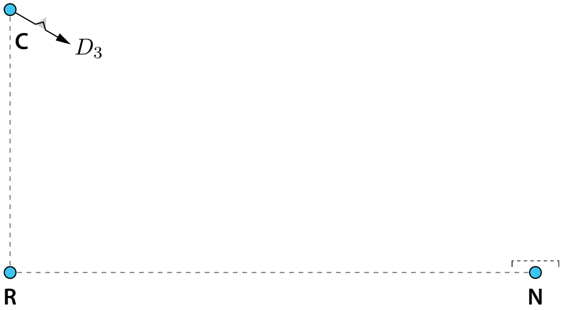
For example, the value x can be mapped to $2d + \dfrac{\lambda}{2} \cdot \dfrac{x}{|x| + 1}$.

Basic component of the TuringMobile



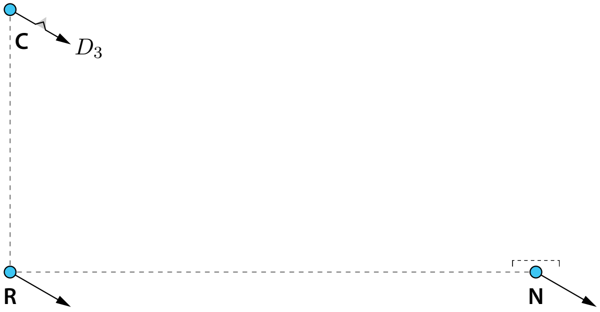
The three robots coordinate themselves to move by a fixed step in one of three fixed directions.

Basic component of the TuringMobile



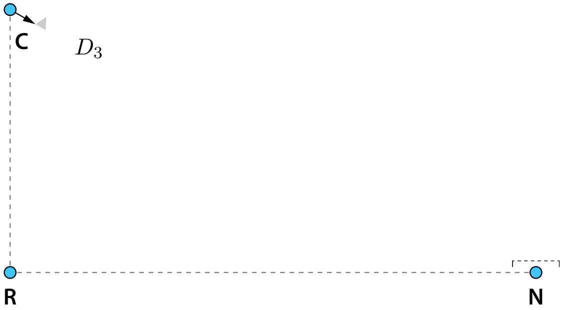
When in the rest position, the Commander chooses its next destination point based also on the state encoded by the Number.

Basic component of the TuringMobile



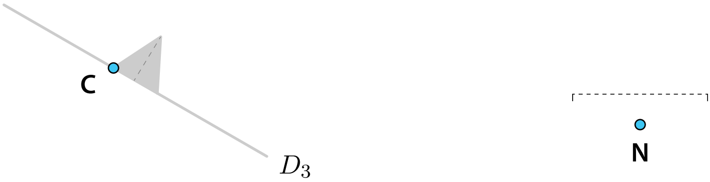
We want all robots to move by the same vector, but it is not wise to let them move at the same time, due to asynchrony.

Basic component of the TuringMobile



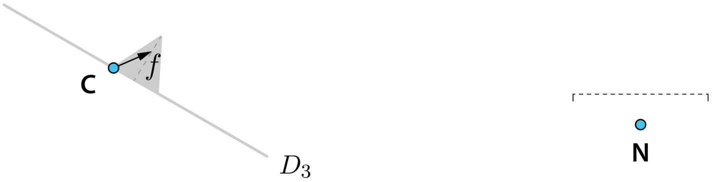
The Commander reaches the middle triangle along its path...

Basic component of the TuringMobile



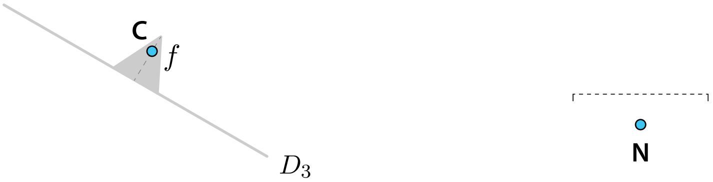
The Commander reaches the middle triangle along its path...

Basic component of the TuringMobile



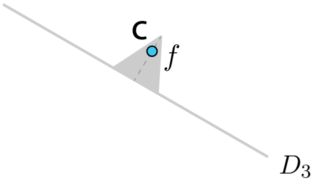
...And it moves to mark the <u>next state</u> of the machine.

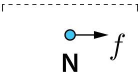
Basic component of the TuringMobile



...And it moves to mark the next state of the machine.

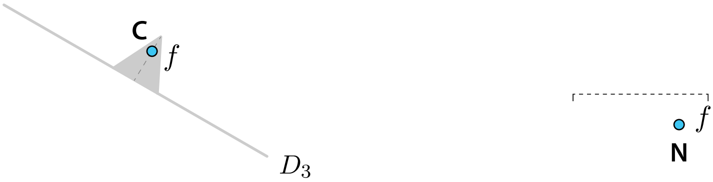
Basic component of the TuringMobile





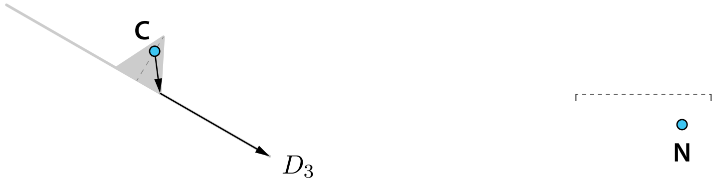
The Number robot sees that and moves to match the same state.

Basic component of the TuringMobile



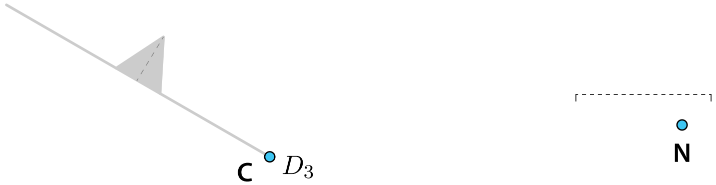
The Number robot sees that and moves to match the same state.

Basic component of the TuringMobile



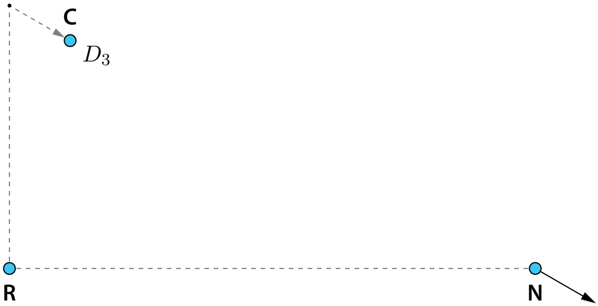
The Commander finishes its move to the destination point.

Basic component of the TuringMobile



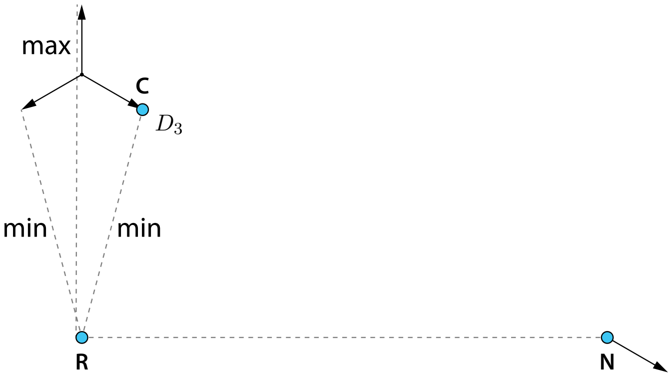
The Commander finishes its move to the destination point.

Basic component of the TuringMobile



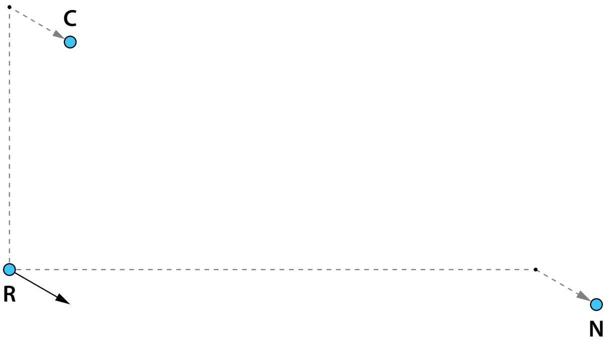
The Number robot moves by the <u>same vector</u> as the Commander.

Basic component of the TuringMobile



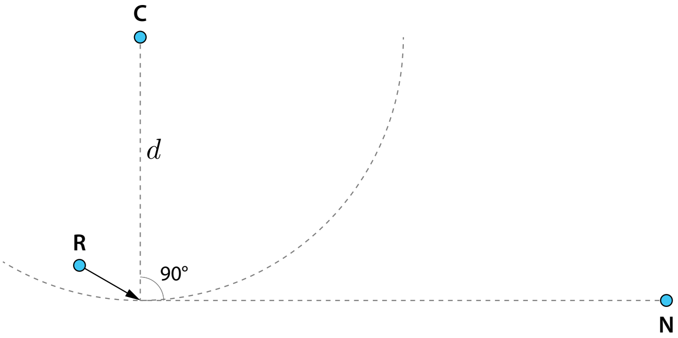
(It starts moving when the distance between the Commander and the Reference is either <u>maximum</u> possible or <u>minimum</u> possible.)

Basic component of the TuringMobile



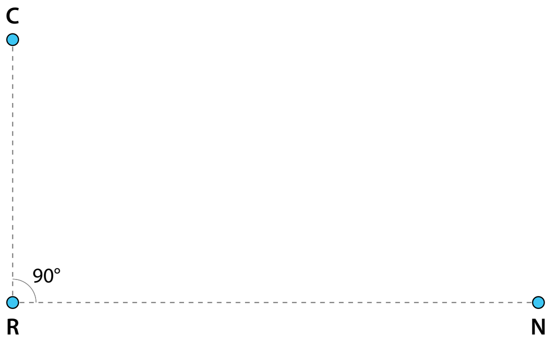
Finally, the Reference robot moves by the same vector, and the system is again in the rest position.

Basic component of the TuringMobile



To determine its destination point, it computes the point at distance d from the Commander that forms an angle of 90°.

Basic component of the TuringMobile



To determine its destination point, it computes the point at distance d from the Commander that forms an angle of 90°.

Basic component: protocol correctness

Does this protocol really work as intended in spite of the robots' asynchrony and unreliability?

Basic component: protocol correctness

Does this protocol really work as intended in spite of the robots' asynchrony and unreliability?

We can decompose the execution into phases:

During each phase, only one robot is supposed to move, while the other two are supposed to wait. *Does this actually happen?*

Basic component: protocol correctness

Does this protocol really work as intended in spite of the robots' asynchrony and unreliability?

We can decompose the execution into phases:

During each phase, only one robot is supposed to move, while the other two are supposed to wait. *Does this actually happen?*

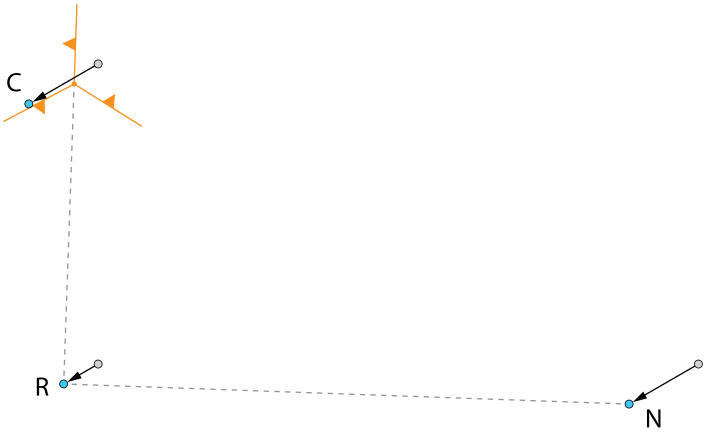
- If a robot r is moving as per phase i and another robot r' sees it (due to asynchrony), we want to prove that r' correctly recognizes the current phase as i, and so it waits.
- If a robot moves as per phase i and is stopped before it reaches its destination (due to unreliability), we want to prove that it takes another snapshot and correctly resumes phase i.

This boils down to showing that no configuration is ambiguous, i.e., it cannot be identified as belonging to two different phases.

Basic component: protocol correctness

Example: When the Reference robot moves, the Commander cannot mistakenly believe that it is its turn to move.

Basic component: protocol correctness



Example: When the Reference robot moves, the Commander cannot mistakenly believe that it is its turn to move.

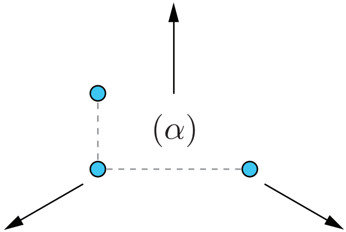
Basic component: protocol correctness

Example: When the Reference robot moves, the Commander cannot mistakenly believe that it is its turn to move.

Basic component: protocol correctness

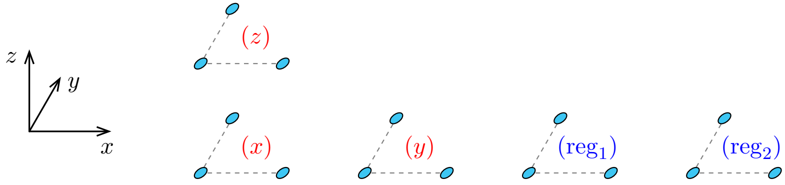
Note: These geometric proofs work because we allow the robots to move in only 3 specific directions.

TuringMobile: full construction



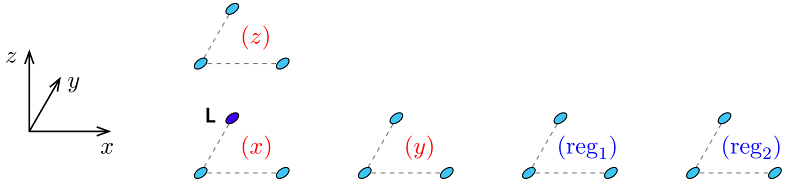
A single basic component can move by fixed-length steps in 3 fixed directions and store and update an arbitrary real number.

TuringMobile: full construction



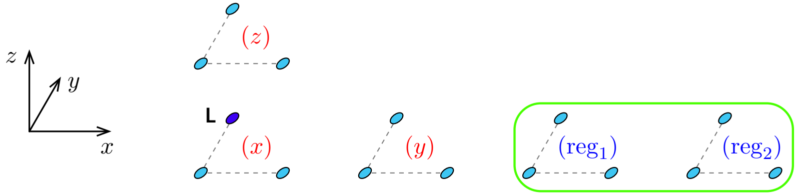
To simulate a reliable robot with k registers in $\mathbb{R}^m$, we use $k + m$ basic components.

TuringMobile: full construction



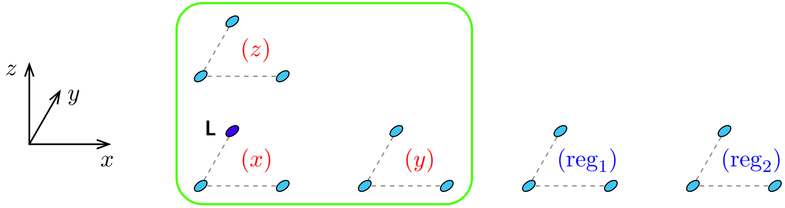
There is an implicit total order on the basic components, and the Commander of the first one is called Leader.

TuringMobile: full construction



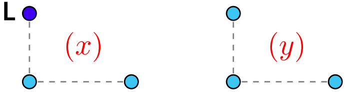
We use k basic components to store the contents of the registers.

TuringMobile: full construction



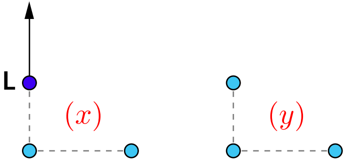
We use m basic components to store the coordinates of the destination point of the TuringMobile with respect to the Leader.

TuringMobile: full construction



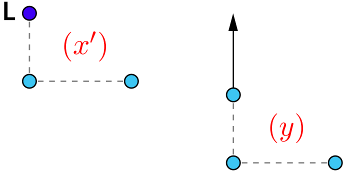
When the TuringMobile has to move, all the components move in order, starting from the Leader's component.

TuringMobile: full construction



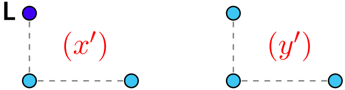
When the TuringMobile has to move, all the components move <u>in order</u>, starting from the Leader's component.

TuringMobile: full construction



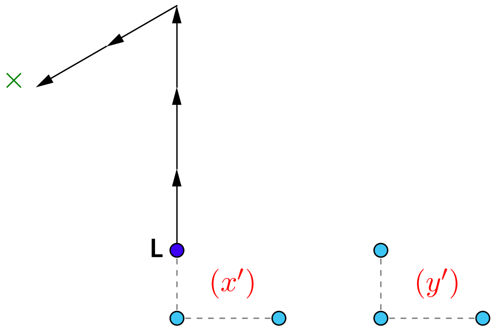
When the TuringMobile has to move, all the components move in order, starting from the Leader's component.

TuringMobile: full construction



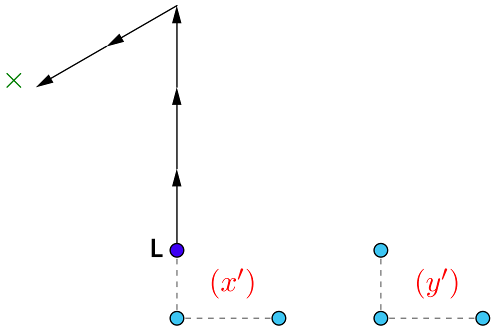
When the TuringMobile has to move, all the components move in order, starting from the Leader's component.

TuringMobile: full construction



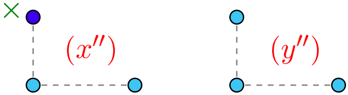
Since the components can only move by fixed-length steps, the TuringMobile may be unable to reach its destination in one step.

TuringMobile: full construction



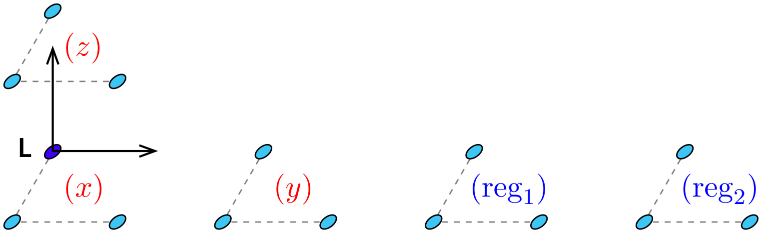
The TuringMobile gets as close as possible to its destination, accordingly updating the values stored in the basic components.

TuringMobile: full construction



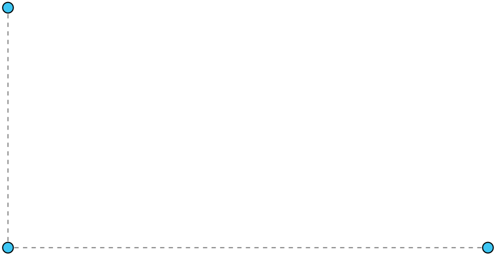
When the TuringMobile cannot get any closer to the destination point, it pretends to be there and computes the next destination.

TuringMobile: full construction



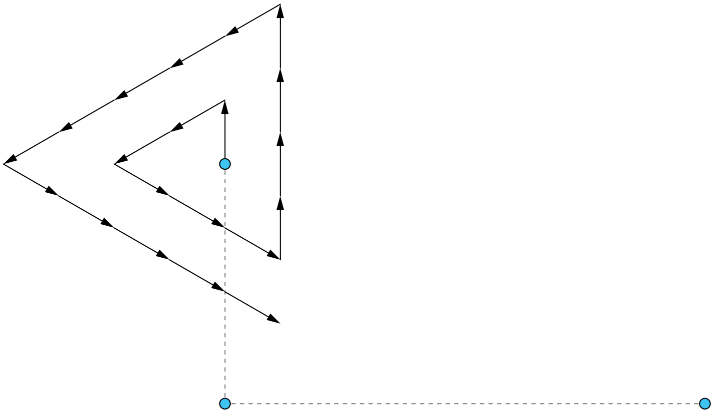
The first m basic components are arranged in space in such a way as to give an m-dimensional sense of direction to the TuringMobile.

Application: exploring the plane



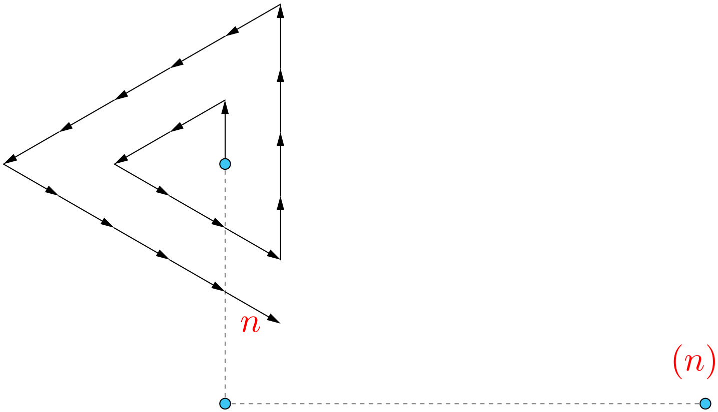
A single basic component can explore the entire plane,
i.e., it can see every point in the course of an infinite execution.

Application: exploring the plane



This is done by performing a spiral-like movement in the three possible directions.

Application: exploring the plane

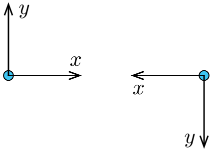


The machine's register counts the number of steps that it has taken, and the Commander reads it to compute its next direction.

Minimality of the basic component

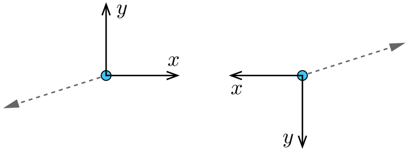
We can indirectly prove that the basic component's design is minimal: indeed, 2 anonymous robots cannot explore the plane.

Minimality of the basic component



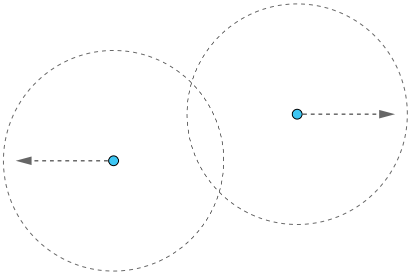
Suppose that their local coordinate systems are oriented symmetrically and they are always activated synchronously.

Minimality of the basic component



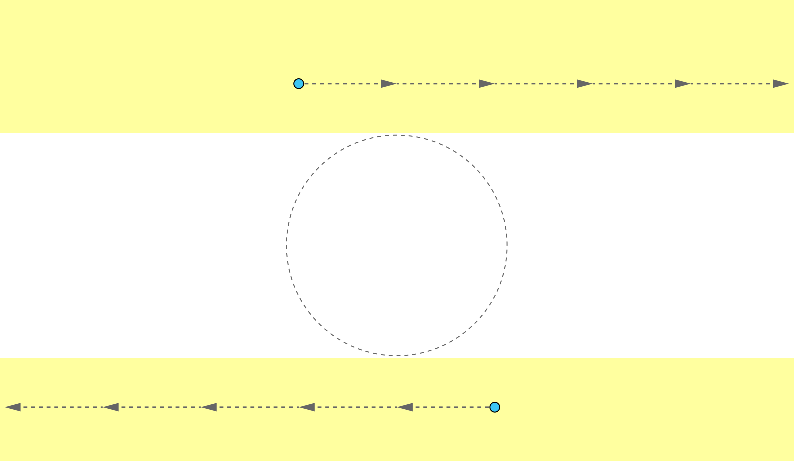
Since they always have symmetric views,
they move in a symmetric way.

Minimality of the basic component



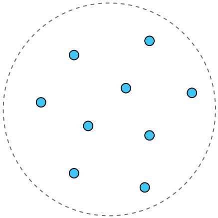
So, in order to explore the plane, they must lose sight of each other. wlog, in this situation they move horizontally.

Minimality of the basic component



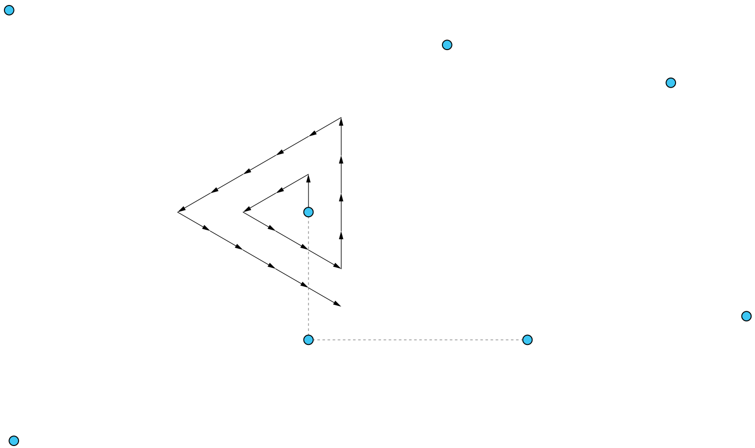
At some point they must stop in the yellow areas. Henceforth, they move horizontally forever, failing to explore the plane.

Application: Near-Gathering with limited visibility



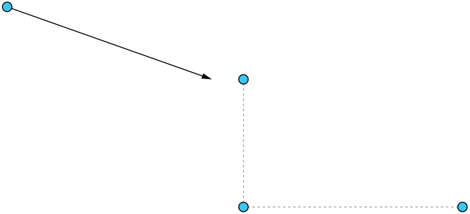
Problem: Make all robots in the plane gather in a small area without ever colliding.

Application: Near-Gathering with limited visibility



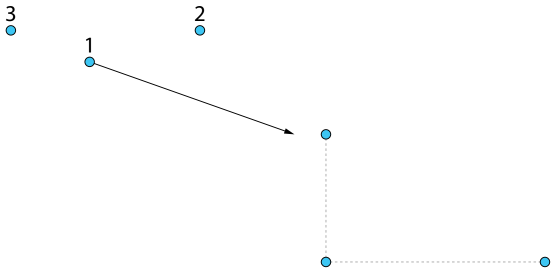
Solution: Put a small-enough basic component of a TuringMobile anywhere, and make it explore the plane.

Application: Near-Gathering with limited visibility



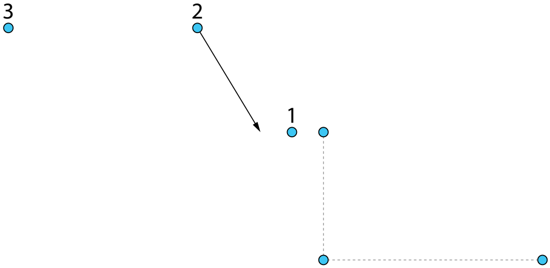
When the TuringMobile gets close enough to a robot, it waits for it to reach a designated area near the Commander.

Application: Near-Gathering with limited visibility



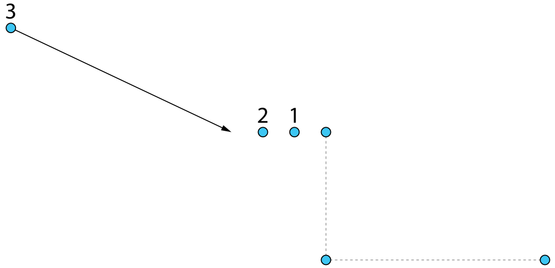
The TuringMobile's shape implicitly gives a total order to the robots that are eligible to move to the designated area.

Application: Near-Gathering with limited visibility



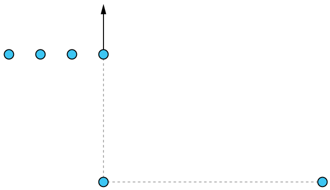
So the robots can coordinate themselves by moving one at a time, avoiding collisions and accidental formation of other TuringMobiles.

Application: Near-Gathering with limited visibility



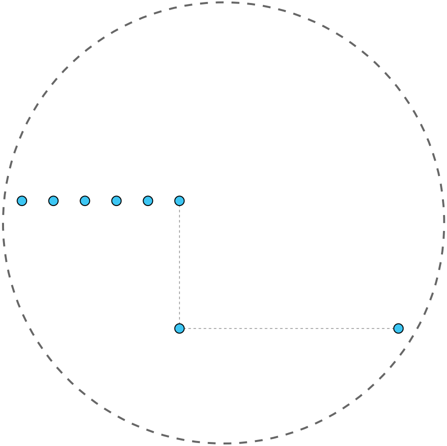
So the robots can coordinate themselves by moving one at a time, avoiding collisions and accidental formation of other TuringMobiles.

Application: Near-Gathering with limited visibility



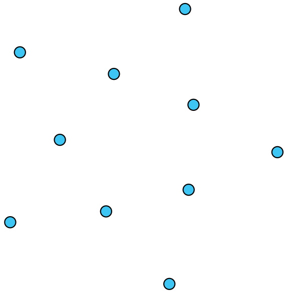
When all the eligible robots have reached the designated area, the TuringMobile resumes the exploration, and the robots follow it.

Application: Near-Gathering with limited visibility

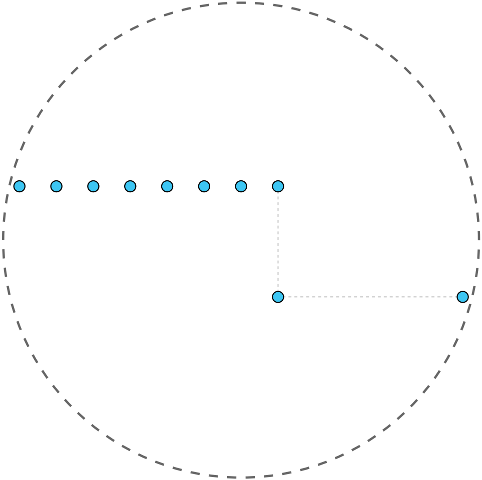


Eventually, all robots are in a small-enough area,
and the Near-Gathering problem is solved.

Application: Pattern Formation with limited visibility

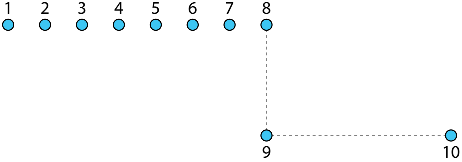


Problem: Make the robots form a given pattern,
arbitrarily rotated or scaled.



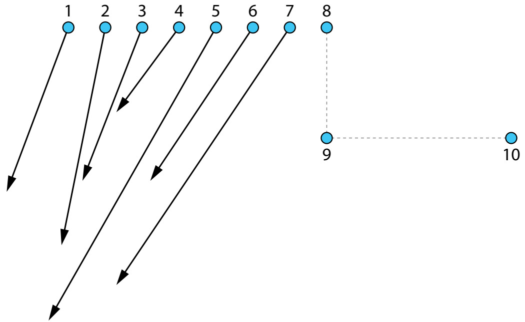
Solution: Solve the Near-Gathering problem first, then form the pattern.

Application: Pattern Formation with limited visibility



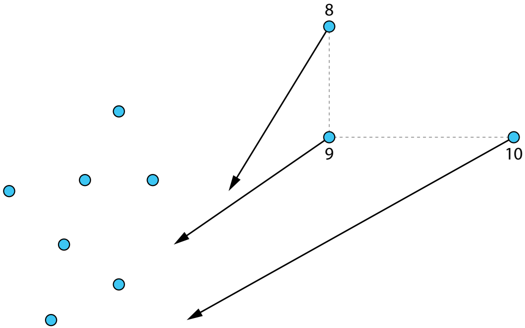
Again, the TuringMobile's shape gives an implicit total order to the robots. We make them move one by one to form the pattern.

Application: Pattern Formation with limited visibility



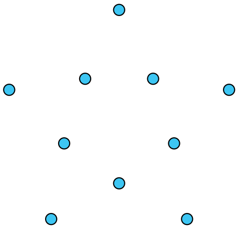
Again, the TuringMobile's shape gives an implicit total order to the robots. We make them move one by one to form the pattern.

Application: Pattern Formation with limited visibility



The last robots to move are the ones constituting the TuringMobile: the others provide a reference to guide them.

Application: Pattern Formation with limited visibility



The last robots to move are the ones constituting the TuringMobile: the others provide a reference to guide them.

Conclusion

Technique: We can simulate a reliable robot with k registers in $\mathbb{R}^m$ with a team of $3k + 3m$ identical unreliable robots with no memory, arranged in a pattern called TuringMobile.

Applications:

- Exploration of the Euclidean space $\mathbb{R}^m$,
- Near-Gathering in $\mathbb{R}^m$ (limited visibility, no axis agreement), provided that a unique TuringMobile is present,
- Pattern Formation in $\mathbb{R}^m$ (limited visibility, no axis agreement), provided that a unique TuringMobile is present.

Open problem: Minimize the number of robots in a TuringMobile.

Conjecture: $k + m + 3$ robots are sufficient.