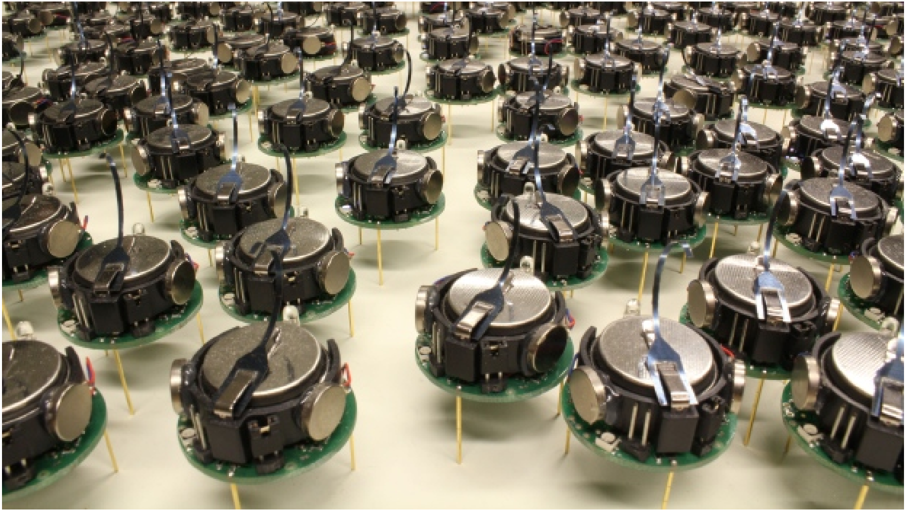
Swarms of Mobile Robots

the Quest for Safety

X. Urbain

Autonomous Mobile Robots



AUTONOMOUS Mobile Robots

Autonomous Mobile Robots

Autonomous Mobile Robots

Autonomous Mobile Robots

Robotic Swarms cooperate to tasks

- Monitoring (wildlife...)
- Ad-hoc networks
- Exploration
- Patrol
- Search and rescue
- ...

Environment $\rightsquigarrow$ harmful ?

THE VERGE TECH SCIENCE CULTURE CARS REVIEWS LONGFORM VIDEO MORE

SCIENCE \ US & WORLD \ TECH

Fukushima's record-high radiation broke a cleaning robot after two hours

Radiation levels are clocking in at 650 Sieverts per hour

by Natt Garun | @nattgarun | Feb 10, 2017, 4:16pm EST

SHARE TWEET LINKEDIN

A robot sent into a Fukushima reactor to inspect and clean the nuclear plant had to abruptly end its mission after excess radiation fried the robot's camera. It was the first time a robot had entered the Unit 2 reactor since the 2011 earthquake and tsunami, reports the *Associated Press*.

NOW TRENDING

7 things we learned from Elon Musk's Tesla shareholder meeting

HP gets in on the external GPU hype with a pretty, large box

Robotic Swarms cooperate to task

- Ad-hoc networks
- Exploration
- Patrol
- Search and rescue
- …

Environment $\rightsquigarrow$ harmful ?

- Low cost — the user can afford to lose some

Robotic Swarms cooperate to task

- Ad-hoc networks
- Exploration
- Patrol
- Search and rescue
- ...

Environment $\rightsquigarrow$ harmful ?

- Low cost — the user can afford to lose some
- Robust protocols — the task can afford to lose some

$\rightsquigarrow$ Weak assumptions — capabilities, silence...

... and **uniform way to describe them**

Robotic Swarms   cooperate to task

- Ad-hoc networks
- Exploration
- Patrol
- Search and rescue
- ...

Lives $\leadsto$ critical !

- Verification/certification

$\leadsto$ **Formal methods**

Certification

Subtle differences + informal reasoning $\leadsto$ error prone

Protocols incorrect w.r.t. specifications still found (see EDCC'15)

Model recent, developping + critical app. $\leadsto$ need formally verified ground

Formal methods

- Model-checking : reachability LTL [Bérard. . ., Devisme. . .]
  + automated – instances (small, discrete)
- Synthesis [Bonnet. . ., Millet. . .]
- Formal proof : proof assistant Coq, Isabelle
  – expertise + scalable, general

$\leadsto$ 2 phases spec/proof : emphasis on ease of specification

Robotic Swarms the need

From a computer science perspective

- Clear characterization, computation model ($\rightsquigarrow$ "realistic")
- Means of verification (matching variants)

$\rightsquigarrow$ Suzuki & Yamashita's model + formal framework

$\rightsquigarrow$ Pactole formal library Courtieu/Rieg/Tixeuil/Urbain

Autonomous Mobile Robots

Suzuki & Yamashita 1999

Robots Move in **Space** according to their **Perception**

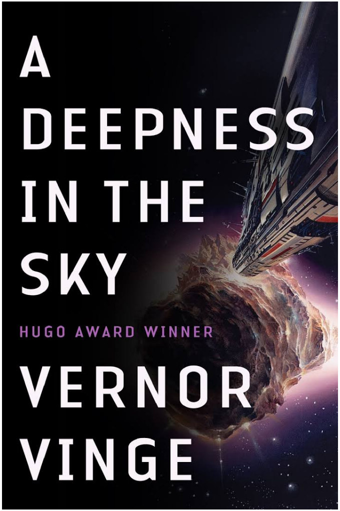
Following :  cycle $\triangle$

Autonomous agents $\rightsquigarrow$ cooperation to realise a task

Without any explicit communication channel

$\rightsquigarrow$ many variants. . .

Autonomous Mobile Robots . . . also in 1999



Spoiler : robotic swarms save the day !

Autonomous Mobile Robots

Suzuki & Yamashita 1999

Robots Move in **Space** according to their **Perception**

Following :  cycle △

Autonomous agents ⇝ cooperation to realise a task

Without any explicit communication channel

⇝ many variants. . . ⇝ "realistic" assumptions

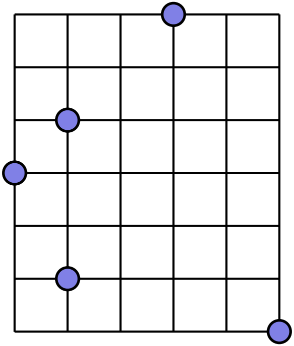
- Space/topology
- Robots' structure and capabilities
- Synchronisation model

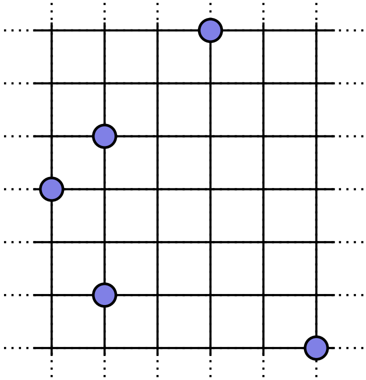
Suzuki & Yamashita 1999

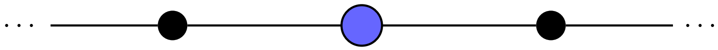
Robots Move in **Space** according to their **Perception**

Characteristics of space

- Topology ?
- Size ?
- . . .

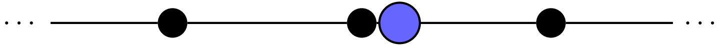


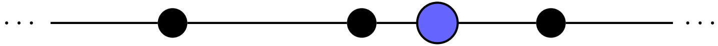


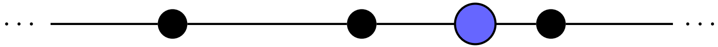


Autonomous Mobile Robots

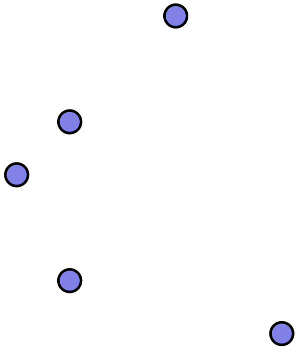
continuous **space**







Autonomous Mobile Robots — continuous **space**



Formalisation **space**

Need for various topologies + generic operations

$\rightsquigarrow$ Space : parameter

Defined as Module `Loc`

- Core : `Loc.t` position in space
- Utils : origin
- Utils : decidable equality
- Utils : distance
- Utils : ...

Instances : graph, $\mathbb{Z}/n\mathbb{Z}$, $\mathbb{R}$, $\mathbb{R}^n$ with relevant arithmetics... that's it

Autonomous Mobile Robots

Suzuki & Yamashita 1999

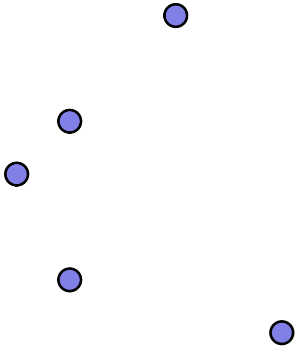
Robots **Move** in **Space** according to their **Perception**

Capabilities of robots

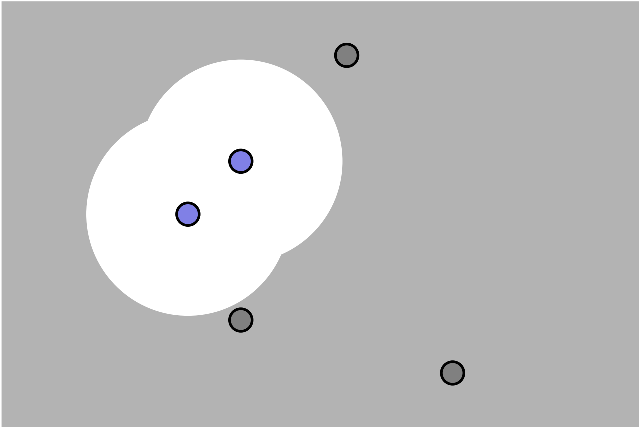
- Memory/Oblivious ?
- Limited/Unlimited perception ?
- What is perceived ? names ? multiplicities ? (“3 here” $\neq$ “Some here”)
- Shared orientation/chirality ?
- Volume ? Energy ? no collision no obstruction ?
- …

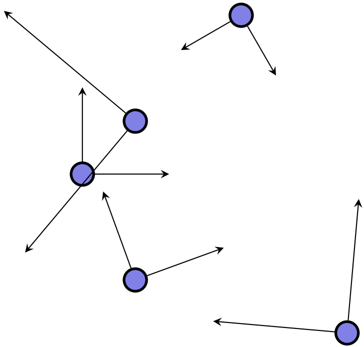
Autonomous Mobile Robots

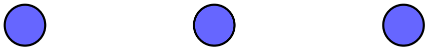
unlimited **vision**



limited **vision**



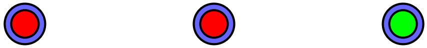




Autonomous Mobile Robots colours

Autonomous Mobile Robots

colours



Autonomous Mobile Robots

Self-visible ?

Formalisation robots, configurations

Robots : set of id., $R = G \uplus B$ size $N = nG + nB$

Conformation : internal state

```
Record Info : Type := (* some state description *) .
Record RobotConf := { loc :> Loc ; robot_info: Info } .
```

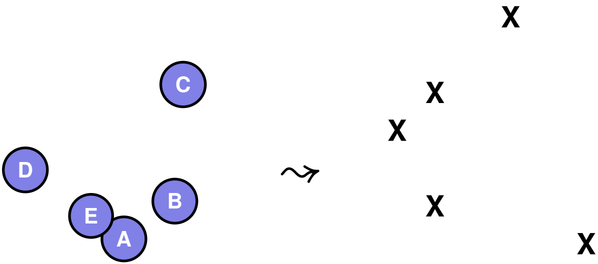
Configuration : display of any robot's conformation simply function

```
Definition configuration := identifier → RobotConf.
```

module with utils : equivalences of config…

Usually too accurate : not the same as perception

Formalisation

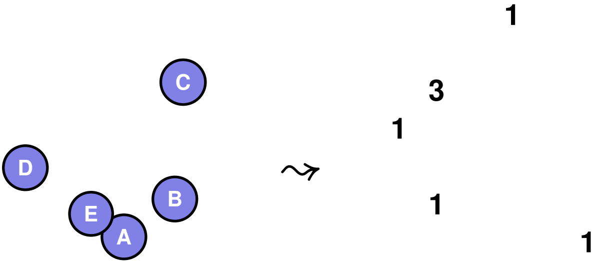


- Roughness of information : observations
- Again a module : equivalence, `from_config`, adequateness. . .

Core : simply function over config → (multi-)sets. . .

Formalisation

perception



- Roughness of information : observations
- Again a module : equivalence, `from_config`, adequateness...

Core : simply function over config → (multi-)sets...

Formalisation embedded program

Algorithm

INPUT : observation wrt r_i

OUTPUT : (conformation with) destination location wrt r_i

Properties

Equivalent perception $\leadsto$ equivalent result

Algorithm + Properties = Robogram

```
Record robogram := {
  pgm :> Obs.t → RobotConf;
  pgm_compat : Proper (Obs.eq ⇒ RobotConf.eq) pgm}.
```

Completely abstract

Higher Order : we can quantify !

Algorithm

INPUT : spectrum wrt r_i

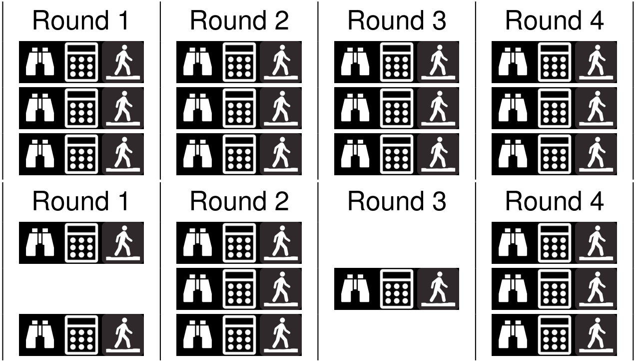
OUTPUT : (conformation with) destination location wrt r_i

```
Definition robogram_pgm (s: Obs.t) : R.t :=
  match Obs.support (Smax s) with (* loc. of max mult. *)
    | nil ⇒ 0                          (* absurd *)
    | pt :: nil ⇒ pt                   (* I: 1 highest tower*)
    | _ ⇒                              (* ... otherwise...  *)
       if beq_nat (length (Obs.support s)) 3 then
       List.nth 1 (sort (Obs.support s)) 0 (*II: mid one *)
       else if is_extremal 0 s then 0    (* ... stay... *)
            else extreme_center s        (* III: center *)
```

Suzuki & Yamashita 1999

Robots **Move** in **Space** according to their **Perception**

Synchronisation and **Movement**



FSYNC

SSYNC

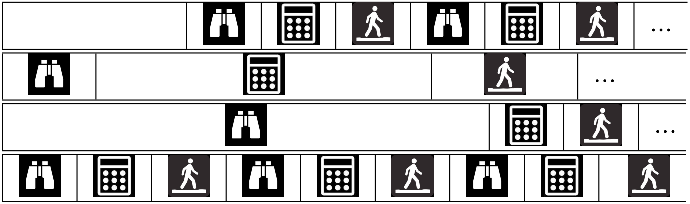
Autonomous Mobile Robots

core

Suzuki & Yamashita 1999

Robots **Move** in **Space** according to their **Perception**

Synchronisation and **Movement**



ASYNC

⇝ outdated perceptions...

$\text{ASYNC} \subseteq \text{SSYNC} \subset \text{ASYNC}^{O(1)}$

Autonomous Mobile Robots core

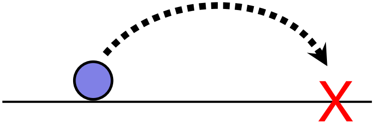
Suzuki & Yamashita 1999

Robots **Move** in **Space** according to their **Perception**

Synchronisation and **Movement**

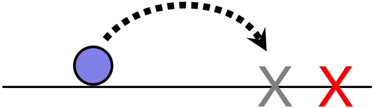
Rigid

X always reached



δ-Flexible

dist $\geq \delta$ if X not reached



Autonomous Mobile Robots

Suzuki & Yamashita 1999

Robots Move in **Space** according to their **Perception**

Core of the model : act of demon (`demonic_action`)

- Selects robots to be activated & assigns changes of state
- Assigns locations/state to Byzantine

F/SSYNC $\rightsquigarrow$ by round, ASYNC $\rightsquigarrow$ by internal state [SSS18 + NETYS19]

Demon

Infinite sequence of its actions : head action, and the rest... (stream)

Round for flexible mvt : new config = new function Proof user only !

```
Definition round δ prot (da: demonic_action) (config) :=
  fun id ⇒
    let loc := config id in (* current loc seen by demon *)
    match da.(step) id with (* is the robot activated? *)
      | None ⇒ loc (* not activated: stay. Never in FSYNC
      | Some (sim, ratio) ⇒ (* new frame, move ratio *)
        let frame_change := sim (config id) in
        let local_config := map frame_change config in
        let local_target := prot (from_config local_config)
        let chosen_target := Loc.mul ratio local_target in
       {| frame_change⁻¹
       (if δ ≤ (Loc.dist (frame_change⁻¹ chosen_target) loc)
           then chosen_target else local_target) ; ... |}
```

Round for flexible mvt : new config = new function Proof user only !

[• • •]

What left for Spec user ?

- Context if not in libraries
- Protocol if needed (correctness)
- Problem / Correctness theorem if not in libraries

Questions :

- What is possible ?
- How it is possible ?
- Is that correct ?

Popular Problems

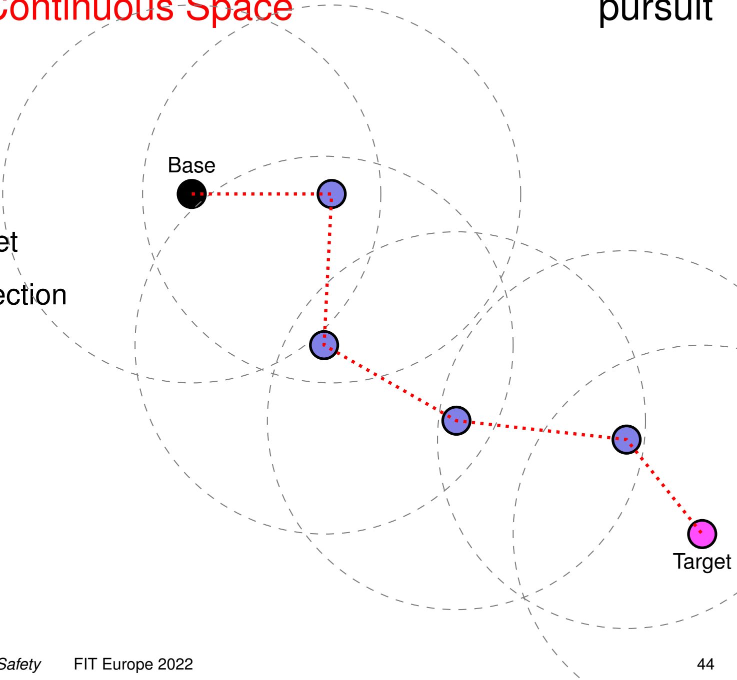
- Probe (patrol/exploration)
- Pursuit (flock/school)
- Pattern formation

Swarms in Continuous Space

Special points :

- Base
- Mobile target

Invariant : connection

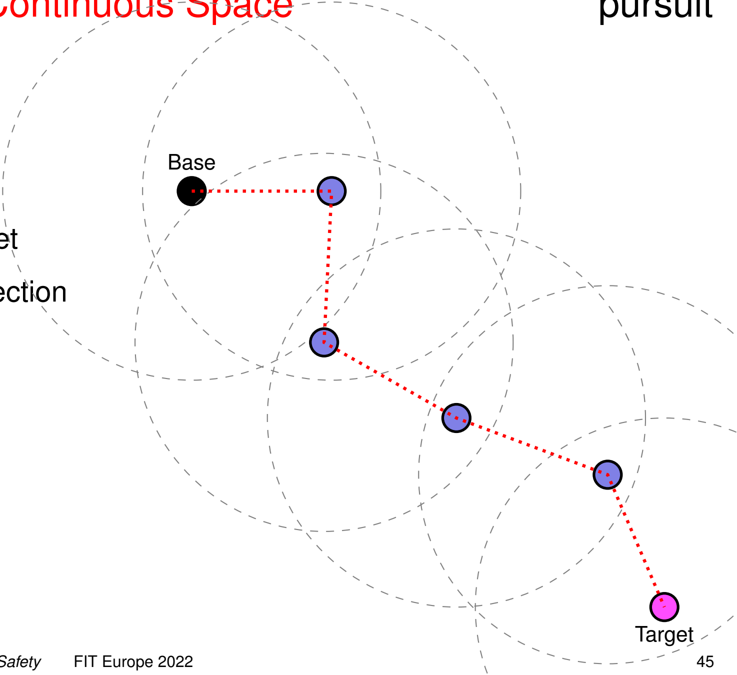


Swarms in Continuous Space

Special points :

- Base
- Mobile target

Invariant : connection

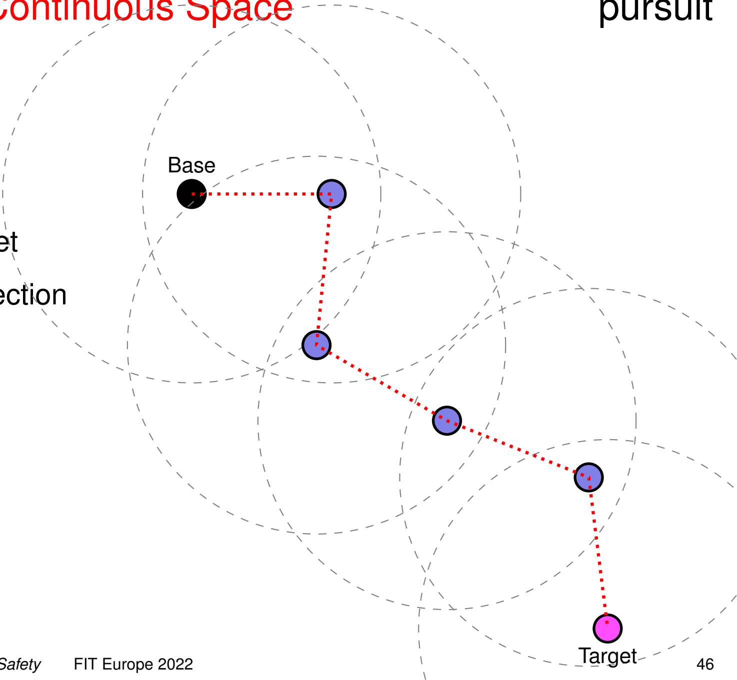


Swarms in Continuous Space

Special points :

- Base
- Mobile target

Invariant : connection

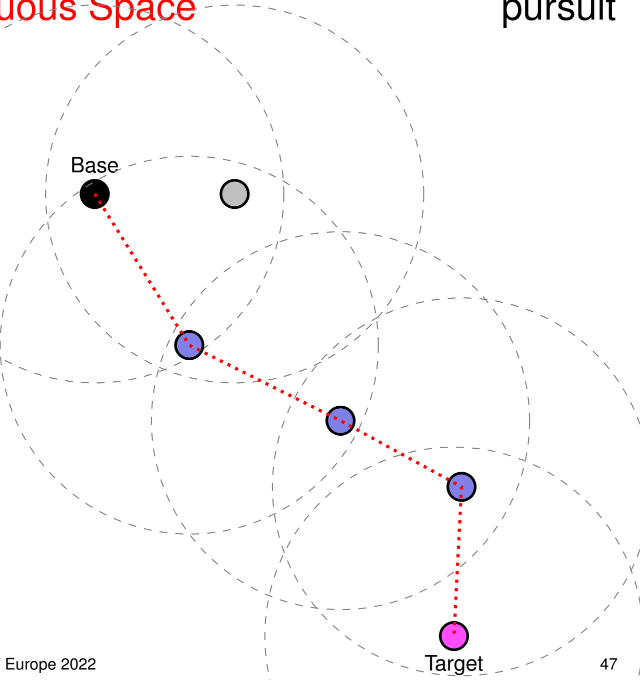


Swarms in Continuous Space

Special points :

- Base
- Mobile target

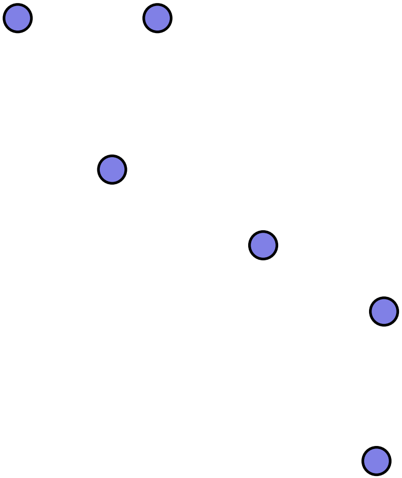
Invariant : connection



Swarms in Continuous Space

Origin : any conf. (scattered)

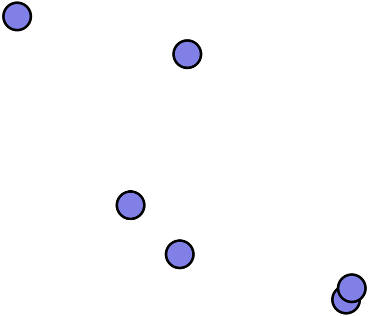
Goal : Arbitrary pattern



Swarms in Continuous Space

Origin : any conf. (scattered)

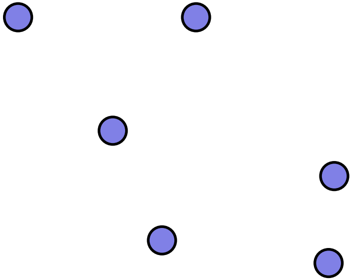
Goal : Arbitrary pattern



Swarms in Continuous Space

Origin : any conf. (scattered)

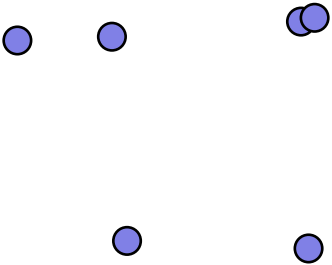
Goal : Arbitrary pattern



Swarms in Continuous Space

Origin : any conf. (scattered)

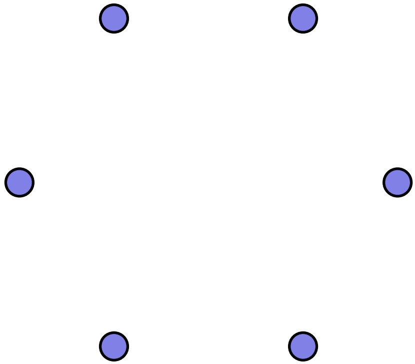
Goal : Arbitrary pattern



Swarms in Continuous Space

Origin : any conf. (scattered)

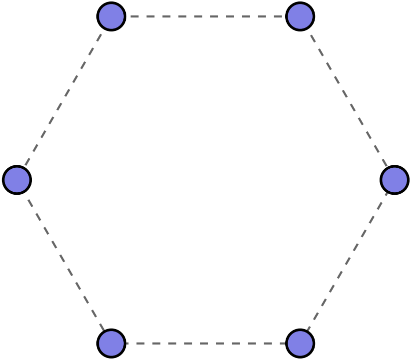
Goal : Arbitrary pattern



Swarms in Continuous Space

Origin : any conf. (scattered)

Goal : Arbitrary pattern



Swarms in Continuous Space

Origin : any conf. (scattered)

Goal : Arbitrary pattern

For $n \geq 3$, APF allows for Leader Election

$\rightsquigarrow$ unsolvable for FSYNC det. oblivious rigid silent robots + chirality [Flocchini et al. 08]

With LE mechanism + chirality $\rightsquigarrow$ solvable in ASYNC [Dieudonné et al. 10]

With full compass $\rightsquigarrow$ solvable in ASYNC [Flocchini et al. 08]

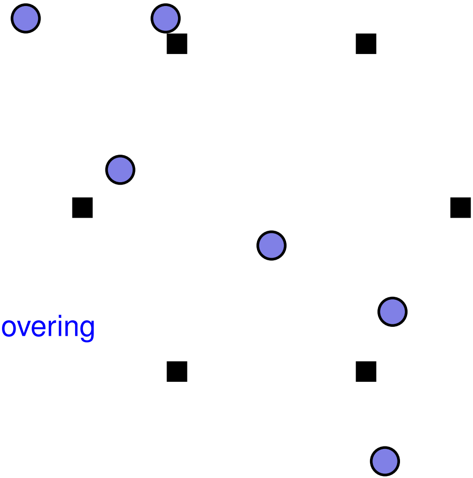
With one axis $\rightsquigarrow$ solvable for odd number in ASYNC

Sequence $\rightsquigarrow$ message...

Swarms in Continuous Space

Origin : any conf. (scattered)

Goal : Arbitrary pattern

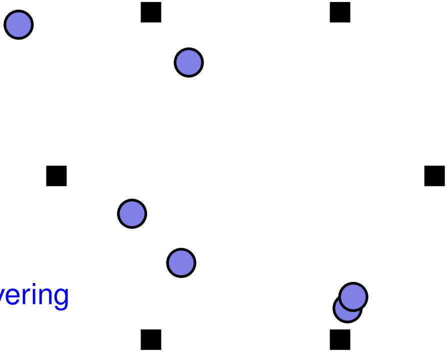


Visible/known $\rightsquigarrow$ Landmark Covering

Swarms in Continuous Space

Origin : any conf. (scattered)

Goal : Arbitrary pattern

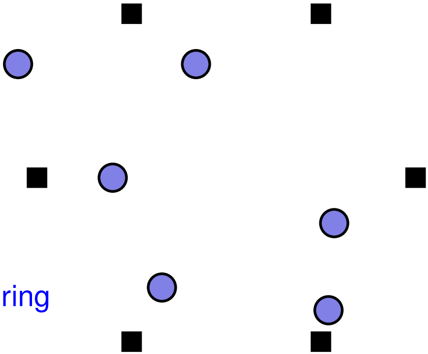


Visible/known ⇝ Landmark Covering

Swarms in Continuous Space

Origin : any conf. (scattered)

Goal : Arbitrary pattern

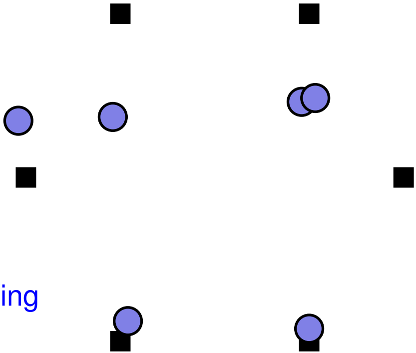


Visible/known ⇝ Landmark Covering

Swarms in Continuous Space

Origin : any conf. (scattered)

Goal : Arbitrary pattern

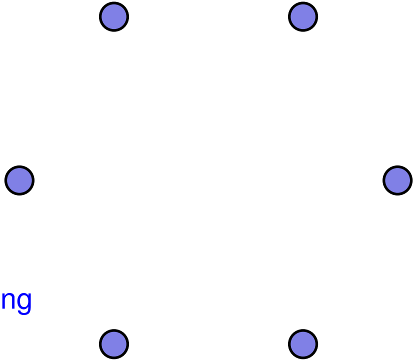


Visible/known ⇝ Landmark Covering

Swarms in Continuous Space

Origin : any conf. (scattered)

Goal : Arbitrary pattern

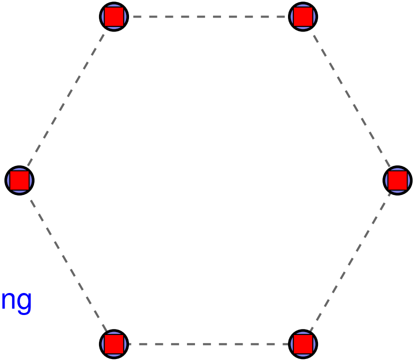


Visible/known $\rightsquigarrow$ Landmark Covering

Swarms in Continuous Space

Origin : any conf. (scattered)

Goal : Arbitrary pattern

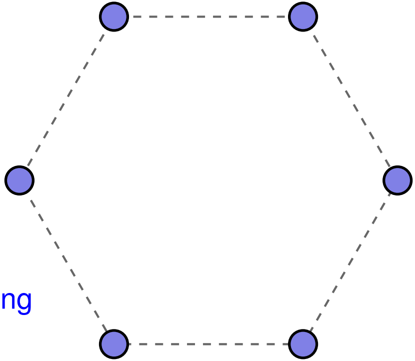


Visible/known ⇝ Landmark Covering

Swarms in Continuous Space

Origin : any conf. (scattered)

Goal : Arbitrary pattern



Visible/known $\leadsto$ Landmark Covering

Swarms in Continuous Space pattern

Origin : any conf. (scattered)

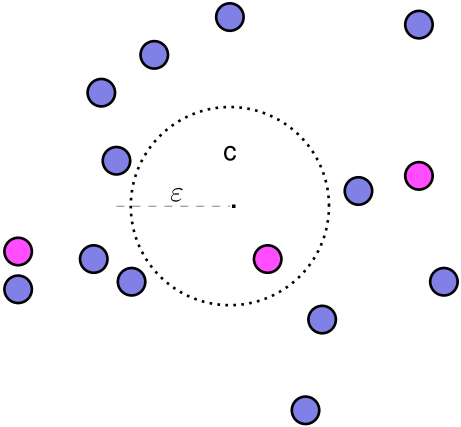
Goal : Arbitrary pattern

With chirality $\rightsquigarrow$ solvable in ASYNC [Fujinata et al. 2010]
“clockwise matching”

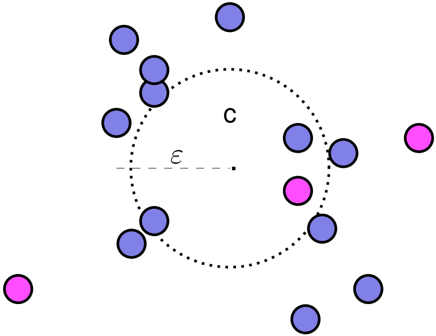
Subcases : benchmarks

- Uniform circle/polygons, *regular*-ish shapes…
- Convergence/Gathering

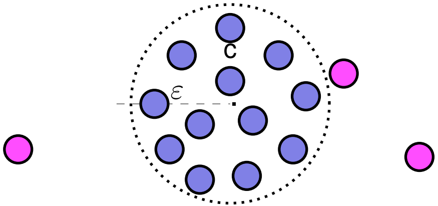
Swarms in Continuous Space

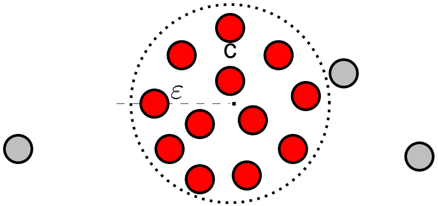


Swarms in Continuous Space



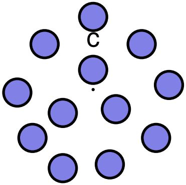
Swarms in Continuous Space



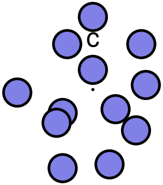


- Forever
  within ε from c
  (coinductive)
- Eventually there
  (inductive)

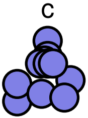
Convergence :
$\exists c, \forall \varepsilon \ldots$


c

Swarms in Continuous Space

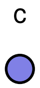


Swarms in Continuous Space



Swarms in Continuous Space

gathering



Swarms in Continuous Space

- Forever at c
  (coinductive)
- Eventually there
  (inductive)

Gathering : $\exists c, \ldots$

Formalisation


```
Definition Gather (pt: Loc.t) (e : execution) :=
Stream.forever (Streams.instant (gathered_at pt)) e.
```

```
Definition WillGather (pt : Loc.t) (e : execution) :=
Stream.eventually (Gather pt) e.
```

Without or with any additional initial condition. . .

```
Definition FullSolGathering (r : robogram) (d : demon) :=
∀ config, ∃ pt : Loc.t, WillGather pt (execute r d config).
```

```
Definition ValidSolGathering (r : robogram) (d : demon) :=
∀ config, ¬invalid config → ∃ pt : Loc.t, WillGather pt...
```

An example

- Anonymous, uniform, points
- Oblivious
- Silent
- No shared orientation

- $\mathbb{R}^2$
- Rigid
- SSYNC

An example

IMPOSSIBILITY of gathering even

Theorem

Gathering impossible for even number of oblivious robots with SSYNC k-fair demon ($k \geq 1$).

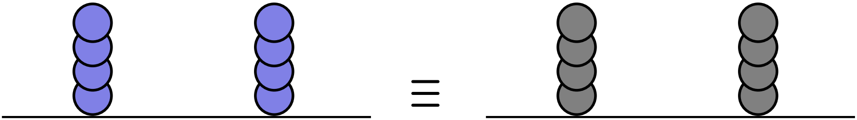
```
Hypothesis even_nG : Nat.Even N.nG.
Hypothesis nG_non_0 : N.nG ≠ 0.

Theorem noGathering:
  ∀ k, (1≤k) → ¬(∀ d, kFair k d → FullSolGathering r d).
```

That's all for specifications... (Proof ~400 lines)

An example

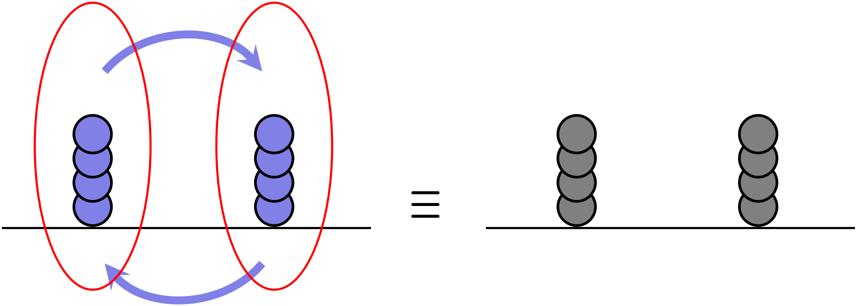
IMPOSSIBILITY of gathering even



An example

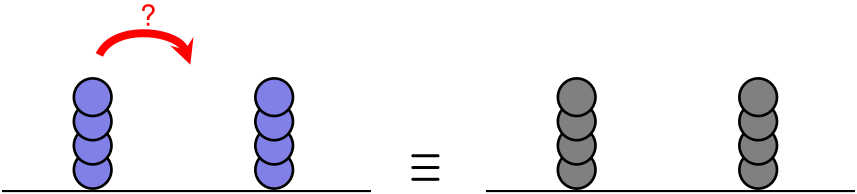
IMPOSSIBILITY of gathering even

An example IMPOSSIBILITY of gathering even



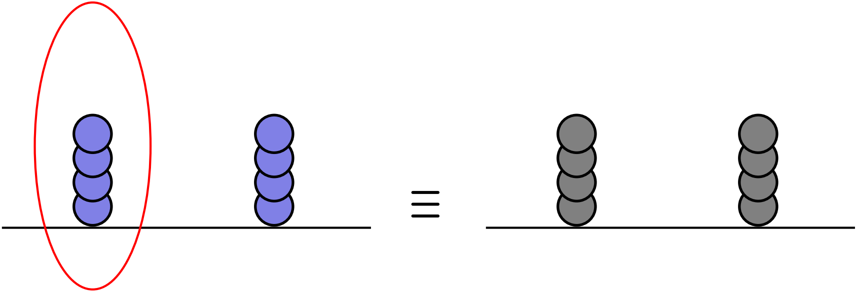
An example

IMPOSSIBILITY of gathering even



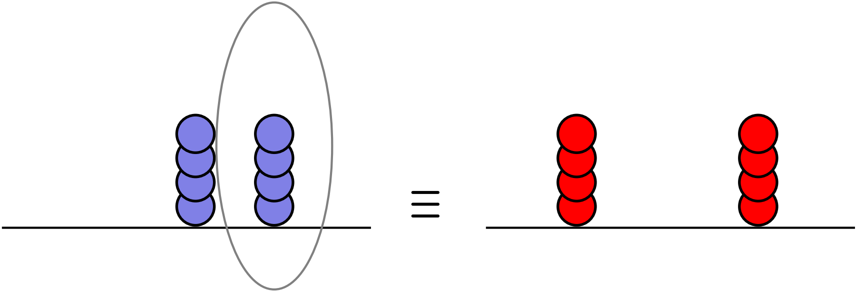
An example

IMPOSSIBILITY of gathering even



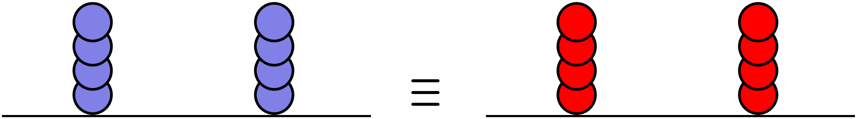
An example

IMPOSSIBILITY of gathering even



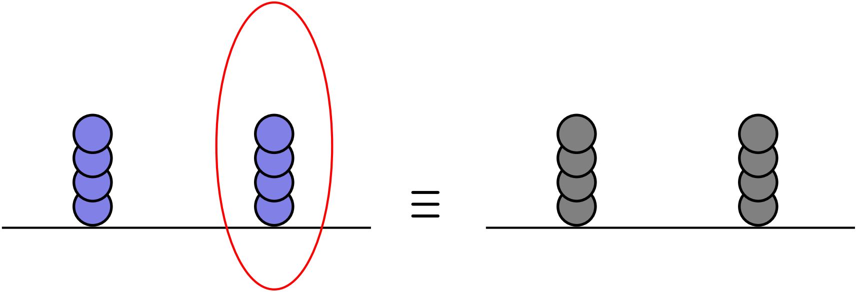
An example

IMPOSSIBILITY of gathering even



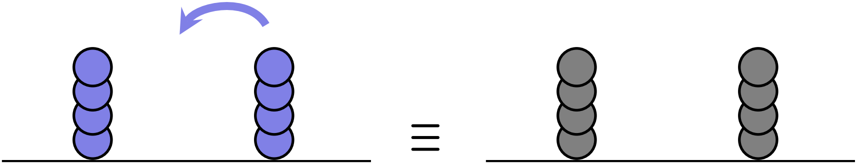
An example

IMPOSSIBILITY of gathering even



An example

IMPOSSIBILITY of gathering even



Robotic Swarms the need

From a computer science perspective

- Clear characterization, computation model ($\leadsto$ “realistic”)
- Means of verification (matching variants)

Robotic Swarms the need

From a computer science perspective

- Clear characterization, computation model ($\leadsto$ “realistic”)
- Means of verification (matching variants)

Robotic Swarms

From a computer science perspective

- Clear characterization, computation model ($\rightsquigarrow$ “realistic”)
- Means of verification (matching variants)

In practice : *Problem oriented* (bounds, optimality)

Robotic Swarms the need

From a computer science perspective

- Clear characterization, computation model ($\leadsto$ "realistic")
- Means of verification (matching variants)

In practice : *Problem oriented* (bounds, optimality)

Fundamental problems Gathering...

Robotic Swarms — the need

From a computer science perspective

- Clear characterization, computation model ($\rightsquigarrow$ “realistic”)
- Means of verification (matching variants)

In practice : *Problem oriented* (bounds, optimality)

Fundamental problems $\rightsquigarrow$Disconnection

Robotic Swarms

the need

From a computer science perspective

- Clear characterization, computation model ($\rightsquigarrow$ “realistic”)
- Means of verification (matching variants)

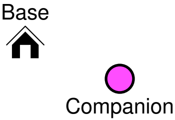
In practice : *Problem oriented* (bounds, optimality)

How to obtain a correct protocol ?

The Life-Line Problem

Given Points :

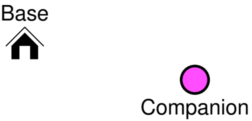
- Base
- Companion



The Life-Line Problem

Given Points :

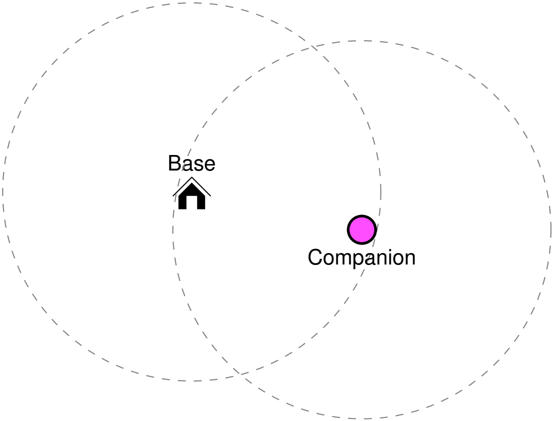
- Base
- Companion



The Life-Line Problem

Given Points :

- Base
- Companion



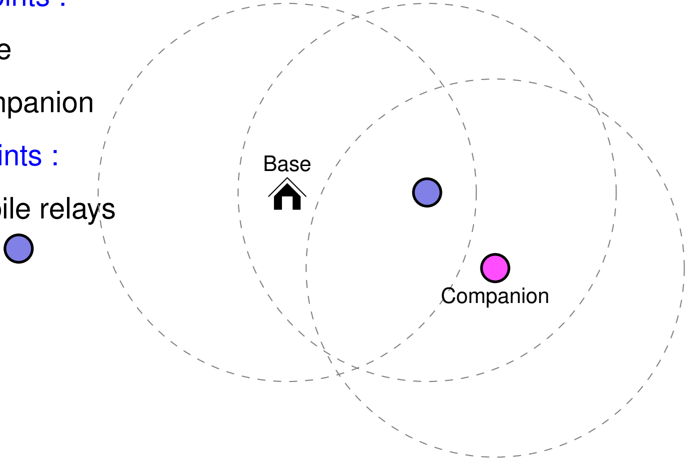
The Life-Line Problem

Given Points :

- Base
- Companion

Extra Points :

- mobile relays



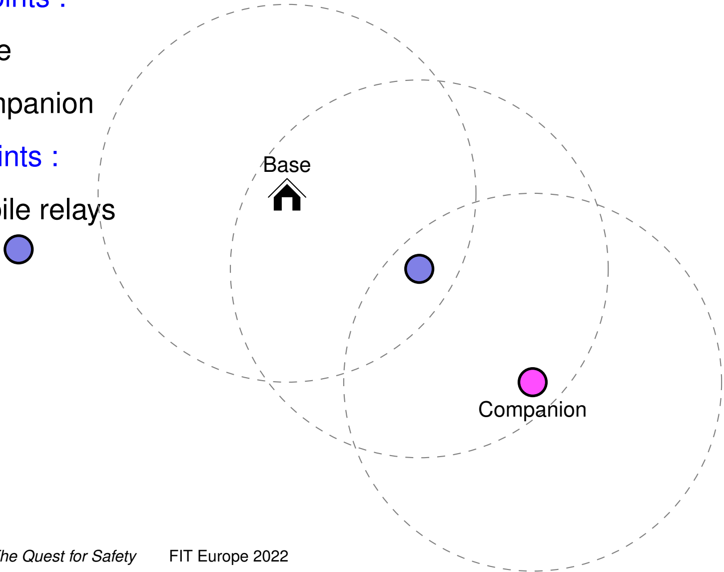
The Life-Line Problem

Given Points :

- Base
- Companion

Extra Points :

- mobile relays



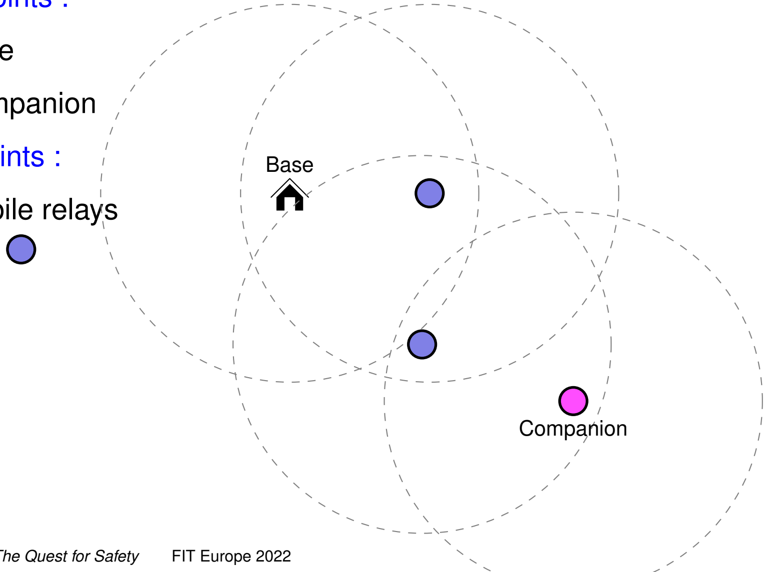
The Life-Line Problem

Given Points :

- Base
- Companion

Extra Points :

- mobile relays



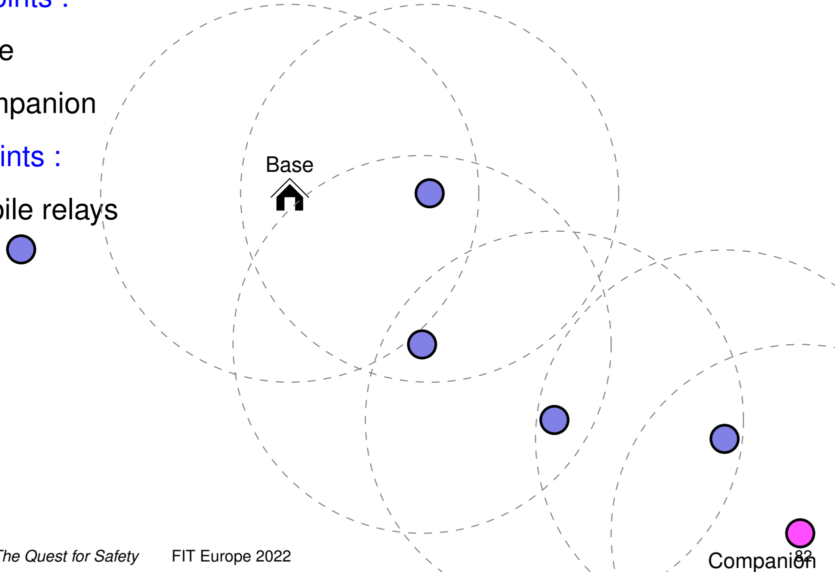
The Life-Line Problem

Given Points :

- Base
- Companion

Extra Points :

- mobile relays



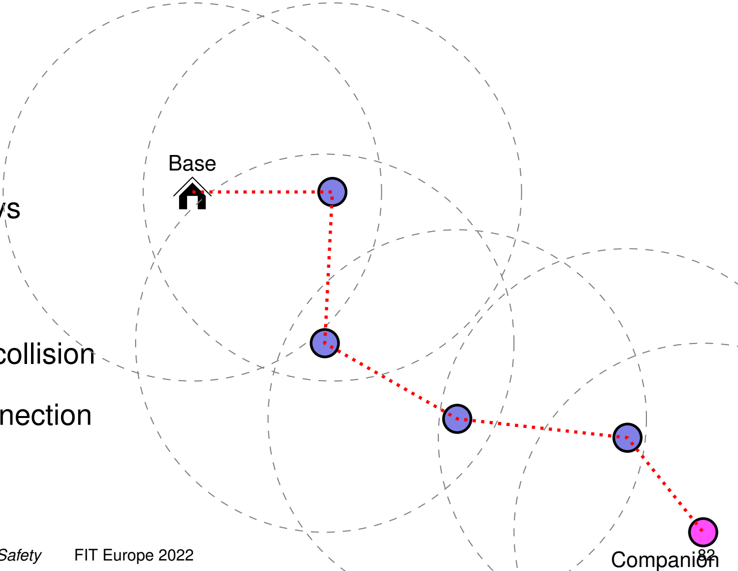
The Life-Line Problem

Given Points :

- Base
- Companion

Extra Points :

- mobile relays



Invariant 1 : no collision

Invariant 2 : connection

The Life-Line Problem

- $\mathbb{R}^3 \rightsquigarrow \mathbb{R}^2$ flying

Companion $\neq$ search team

- Special location : base

The Life-Line Problem

- $\mathbb{R}^3 \rightsquigarrow \mathbb{R}^2$ flying — Companion ≠ search team
- Special location : base

- No detection of multiplicity (volume/collision)
- Vision limitée — D_{max}
- Vitesse limitée
- No Byzantine

The Life-Line Problem

- $\mathbb{R}^3 \rightsquigarrow \mathbb{R}^2$ flying — Companion ≠ search team
- Special location : base
- No detection of multiplicity (volume/collision)
- Vision limitée — D_{max}
- Vitesse limitée
- No Byzantine
- FSYNC
- Rigid mvt
- Start from base

The Life-Line Problem

Invariants formal statement

No collision

```
∀ r r′, r ≠ r′ →location(r) ≠ location(r′)
```

Connected path

```
∀ r, alive(r) →
          (∃ r′, alive(r′)  ∧  ident(r) > ident(r′)
               ∧  dist(location(r), location(r′)) ≤ D_max)
          ∨  r = companion
```

The Life-Line Problem

Iterative refinement :

- which robot to follow ? identifiers (scout = 0)

The Life-Line Problem

Iterative refinement :

- which robot to follow ? identifiers (scout = 0)
- avoiding collisions robot elimination (alive/dead, defensive)

The Life-Line Problem

Iterative refinement :

- which robot to follow ? — identifiers (scout = 0)
- avoiding collisions — robot elimination (alive/dead, defensive)
- connection despite elimination — communication with lights

What could be a **correct** protocol candidate

1. Chose a target robot (direction)
2. Chose a destination
3. Safe to go ?
   - Yes : go to destination, no warning
   - No : stay, warn of danger

What could be a **correct** protocol candidate

1. Chose a target robot (direction)
2. Chose a destination
3. Safe to go ?
   - Yes : go to destination, no warning
   - No : stay, warn of danger

```
Definition protocole (s : observation) : R2*light :=
 let target := choose_target s in
 let new_pos := choose_new_pos s (fst target) in
 match move_to s new_pos with      (* Is this dangerous? *)
 | true ⇒ (new_pos,false)    (* Safe: move + light off. *)
 | false ⇒ ((0,0), true)  (* Danger: stay + light on.  *)
 end.
```

What could be a **correct** protocol candidate

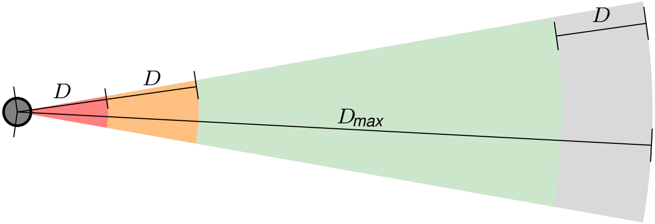
Constraints on

- Target
- Destination

```
Definition protocole (s : observation) : R2*light :=
 let target := choose_target s in
 let new_pos := choose_new_pos s (fst target) in
 match move_to s new_pos with      (* Is this dangerous? *)
 | true ⇒ (new_pos,false)      (* Safe: move + light off. *)
 | false ⇒ ((0,0), true)    (* Danger: stay + light on.  *)
 end.
```

What could be a **correct** protocol candidate

```
Axiom choose_new_pos_spec : ∀ obs target,
  let new := choose_new_pos obs target in
    dist new target ≤ Dp
    ∧ dist new (0,0) ≤ D.
```



What could be a **correct** protocol candidate

```
Axiom choose_target_spec : ∀ obs_id local_config,
   let obs := obs_from_config local_config in
   let target := choose_target obs_id obs in
      target ∈ obs                     (* target must be in range *)
    ∧ get_alive target = true                         (* be alive *)
    ∧ get_ident target < get_ident obs_id (* smaller id *)
    ∧ (get_light target = true  (* preferably light off *)
         → ∀ id ∈ obs, get_light id = true)
    ∧ (get_light target = true       (* preferably close *)
         → dist (0,0) (get_loc target) > Dp
         → ∀ id ∈ obs, dist (0,0) (get_loc elt) > Dp).
```

What could be a **correct** protocol family

A family of correct protocols

Theorem. Both invariants hold along any FSYNC execution from legit initial configuration for a candidate fulfilling constraints.

Specifications and proof : 1500 lines of Coq

Solution obtained inside formal framework along specifications

Additional proof : family not empty (easy)

What could be a **correct** protocol family

Actual formal statement of invariants

```
Definition path_conf (cf:config) :=
∀ g, get_alive (cf g) = true →
 get_ident (cf g) = 0
  ∨ ∃ g′, dist (get_loc (cf g)) (get_loc (cf g′)) ≤ Dmax
               ∧ get_alive (cf g′) = true
               ∧ get_launched (cf g′) = true
               ∧ get_ident (cf g′) < get_ident (cf g).
```

```
Definition no_collision_conf(cf:config) := ∀ g g′, g ≠ g′
  → get_launched(cf g) = true → get_launched(cf g′) = true
  → get_alive(cf g) = true → get_alive(cf g′) = true
  → dist (get_loc (cf g)) (get_loc (cf g′)) ≠ 0ℝ.
```

```
Definition NoCollAndPath e :=
forever (fun c ⇒ no_collision_conf c ∧ path_conf c) e.
```

A few words on Pactole

Pactole = formal library for the Coq proof assistant
modelling robotic swarms https://pactole.liris.cnrs.fr/

Assets :

- single framework to express everything
- specification is easy (very close to math)
- proof of correctness + of impossibility
- compare expressive power of models

Examples :

- Space : ring, graph, plane, etc.
- Problems : gathering, convergence, exploration, lifeline