Communicating systems
https://www.ensta-bretagne.fr/jaulin/comsys.html

Luc Jaulin

February 25, 2025

Chapter 1

Logic programming

Logic programming [1] is a type of programming paradigm associated with artificial intelligence. It is based on formal logic.

Any program written in a logic programming language is a set of sentences in logical form, expressing facts and rules [2][3].

In this lesson, we will use GNU PROLOG (also called GPROLOG) which is a compiler which supports some extensions to PROLOG including constraint programming over a finite domain. The compiler converts the source code into byte code that can be interpreted by a Warren abstract machine (WAM). Similarly, we can use Prolog online : https://swish.swi-prolog.org/ which does not need any install.

A Horn clause is defined as

$$h \leftarrow b_1 \wedge b_2 \wedge \cdots \wedge b_n$$

where h is called the *head* of the rule and $b_1, \ldots, b_n$ is called the *body*. Clauses with empty bodies (*i.e.*, $n = 0$) are called *facts*. If $n > 1$ the clause is a *rule*.

In PROLOG, we write

```
h :- b1, b2, ..., bn
```

EXERCISE 1.– *Sibling*

We have 5 persons: a,b,c,d,e linked by family relations such as mother, father, sibling described by the following clauses (given with the PROLOG syntax).

```
mother(a,b).
father(c,b).
father(c,d).
father(e,c).
parent(X,Y) :- father(X,Y).
parent(X,Y) :- mother(X,Y).
sibling(X,Y) :- parent(Z,X), parent(Z,Y), X\=Y.
```

1) Using PROLOG ask for the following queries:

```
?- sibling(X,d).
?- father(X,d).
?- mother(X,d).
```

2) Show that this problem is a *constraint satisfaction problem*. Give the atoms, the variables and the constraints.

3) Using the rules of logic, show how the resolution can be done.

EXERCISE 2.– *Hanoi towers*

We have three pegs: left, center, right. We have to move N disks from the left peg to the right peg using the center peg as an auxiliary holding peg. At no time can a larger disk be placed upon a smaller disk.

In what follows, move(N,X,Y, Z) is true if we can move the N top disks from the peg X to the peg Y using the peg Z

```
move(1,X,Y,_) :-
  write('Move from '), write(X), write(' to '), write(Y), nl.
move(N,X,Y,Z) :-
  N>1,
  M is N-1,
  move(M,X,Z,Y),
  move(1,X,Y,Z),
  move(M,Z,Y,X).
```

Using PROLOG, which query should we enter to move $N = 3$ disks.

EXERCISE 3.– *N-queens puzzle*

The challenge is to set N queens on an $N \times N$ grid so that no queen can "take" any other queen. Queens can move horizontally, vertically, or along a $\pm\frac{\pi}{4}$ diagonal. The following matrix shows a solution for $N = 4$ queens.

$$\mathbf{M} = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{pmatrix}$$

A solution to this puzzle can be represented as a special permutation of the list [1,2,3,4]. For example, $\mathbf{M}$ can be represented by the vector $J = [3, 1, 4, 2]$. This representation prevents two or more queens in the same row or column.

To test whether a given permutation is a solution, one needs to calculate whether the permutation has two or more queens on the same diagonal. Two queens at position (i_1, j_1) and (i_2, j_2) are on the same diagonal

$$i_1 + j_1 = i_2 + j_2 \text{ or } i_1 - j_1 = i_2 - j_2$$

For the kth queen at position (i_k, j_k), we associate the two quantities

$$\begin{aligned} s_k &= i_k + j_k \\ d_k &= i_k - j_k \end{aligned}$$

To check that a configuration of queens is diagonal consistent, it suffices to check the vector $\mathbf{s} = (s_1, \ldots, s_n)$ and the vector $\mathbf{d} = (d_1, \ldots, d_n)$ both satisfy the alldiff constraint.

We define the constraint $\text{sum}(I, J, S)$ as follows

$$\text{sum}(I, J, S) \Leftrightarrow \forall k, i_k + j_k = s_k$$

and the constraint $\text{diff}(I, J, D)$ as follows

$$\text{diff}(I, J, D) \Leftrightarrow \forall k, i_k - j_k = d_k$$

For instance, for the matrix $\mathbf{M}$ given above, we have

$$\begin{aligned} I &= [1, 2, 3, 4] \\ J &= [3, 1, 4, 2] \end{aligned}$$

To satisfy sum constraint, we should take :

$$S = [4, 3, 7, 6]$$

To satisfy diff constraint we should take :

$$D = [-2, 1, -1, 2]$$

1) Using PROLOG, define the sum and diff constraint
2) Solve the queens puzzle for an 8x8 board.
3) Solve the queens puzzle for an 8x8 board, is the case where we ask one queen at top-left corner.

EXERCISE 4.– *Who owns the zebra?*

Houses logical puzzle: who owns the zebra and who drinks water?
1) Five colored houses in a row, each with an owner, a pet, fruit, and a drink.
2) Bob lives in the red house.
3) Eve has a dog.
4) They drink coffee in the green house.
5) Alice drinks tea.
6) The green house is next to the white house.
7) The grape eater has a snake.
8) In the yellow house they eat apple.
9) In the middle house they drink milk.
10) David lives in the first house from the left.
11) The papaya eater lives near the house with the fox.
12) In the house next to the house with the horse they eat apple.
13) The mango eater drinks beer.
14) Carol eats banana.
15) David lives near the blue house.
16) In one of the house, they drink water.
17) Zebra is one of the pet.

To have a complete view of the puzzle, we will use a table Hs, where each line corresponds to a house.

House	**Owner**	**Pet**	**Fruit**	**Drink**	**Color**
1	David				
2					
3				Milk	
4					
5					

Equivalently, each line can be represented by the predicate:

```
h(Name,Pet,Fruit,Drink,Color)
```

which is true if the corresponding entries form a line of the table

Using PROLOG, find who owns the zebra and who drinks water?

Chapter 2

Dynamic Epistemic Logic

Dynamic epistemic logic [4] is a logical framework dealing with knowledge and information change. It focuses on situations involving multiple agents (or robots) and studies how their knowledge changes when events occur. These events can change factual properties of the actual world (they are called ontic events): for example the robot A is moving. They can also bring about changes of knowledge without changing factual properties of the world (they are called epistemic events): for example the robot B has recognized an object it has already seen.

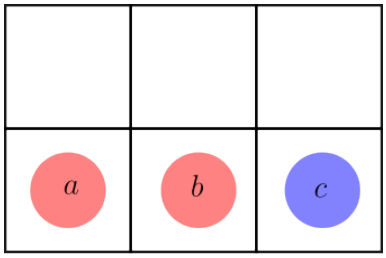
Epistemic logic is a modal logic dealing with the notions of knowledge and belief. As a logic, it is concerned with understanding the process of reasoning about knowledge and belief. A classical example illustrating dynamic epistemic logic is the *muddy children problem,* given in the following section.

2.1 Muddy children problem

We consider three robots a, b, c, that are either blue or red. Here, we replaced *children* and *muddy* of the original *muddy children problem* by robots and colors to adapt the puzzle to robotics. Assume that

- the robots can see each other.
- the robots do not know their own color.
- The robots cannot communicate

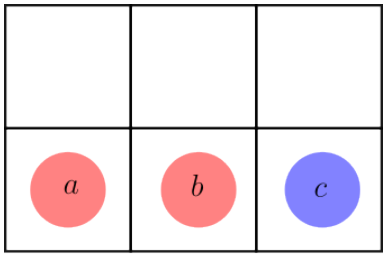
We consider the situation of the Figure 2.1.



$k = 0$

Figure 2.1: Three robots (2 red, 1 blue) who do not know their color.
The robots can see each other and cannot communicate

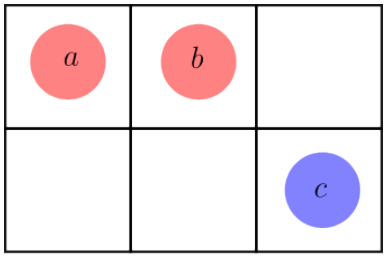
We assume that God says to all robots: "At least one of you is red. If you know you are red : go up". Here, God plays the role of someone any robot obeys. At time $k = 1$, nothing has changed since no robot knows its own color (see Figure 2.2). But this only valid for the ontic state. Indeed, the knowledge of the robots has changed: they have observed the behavior of the other robots. From this, the robots might be able to deduce their own color.



$k = 1$

Figure 2.2: No robot has moved, but they have observed the behavior of others

Indeed, at time $k = 2$, the two red robots moved up, as illustrated by Figure 2.3. Indeed, for $k = 1$, the two red robots were able to find their color.



$k = 2$

Figure 2.3: The two red robots moved up

We have two different types of states

- The color of the robots is the ontic state
- Their memory is the epistemic state

To understand and formalize the distributed reasoning made by the robots, we define the individual knowledge operator K_i for agent i.

Individual knowledge

The sentence $K_i\phi$ means that the robot i, knows ϕ. It is an individual knowledge which is valid for a specific robot. For instance, for our problem, at time $k = 0$

(i) $K_a K_b \cdot \text{color}(c) = \text{blue}$
(ii) $K_c \left(K_b \cdot \text{color}(c) = \text{blue} \vee K_b \cdot \text{color}(c) = \text{red}\right)$

And at time $k = 1$, we have
(i) $K_b \cdot \text{color}(b) = \text{red}$
(ii) $\neg K_c K_b \cdot \text{color}(b) = \text{red}$
(iii) $\neg K_c \cdot \text{color}(c) = \text{blue}$
We can understand that we have properties or rules such as

$$K_1 K_2 \cdot \phi \Rightarrow K_1 \cdot \phi$$

General knowledge

We define the general knowledge operator as

$$E \cdot \phi \Leftrightarrow \bigwedge_i K_i \cdot \phi$$

For instance, for our problem, at time $k = 2$, we have

(i) $E \cdot \text{color}(a) = \text{red}$
(ii) $E \cdot \text{color}(c) = \text{blue}$

Common knowledge

We define the common knowledge operator

$$C \cdot \phi \Leftrightarrow \bigwedge_k E^k \cdot \phi$$

The common knowledge has to be understood as a proposition which is true and which is written on a blackboard everybody can see.

Resolution

To solve our problem, we use the following reasoning.

- at $k = 0$: we have $K_a\left(Blue(a) \Rightarrow Step(b)\right)$
- at $k = 1$, we have

$$\begin{array}{ll} & \neg Step(b) \\ \Rightarrow & K_a\left(\neg Blue(a)\right) \\ \Rightarrow & K_a\left(Red(a)\right) \\ \Rightarrow & Step(a) \end{array}$$

- Therefore, at $k = 2$, the two red robots went up.

Look-compute-move model

From this example, we see that some interactions between robots may occur, without any communication. It corresponds to the look-compute-move model illustrated by Figure 2.4.

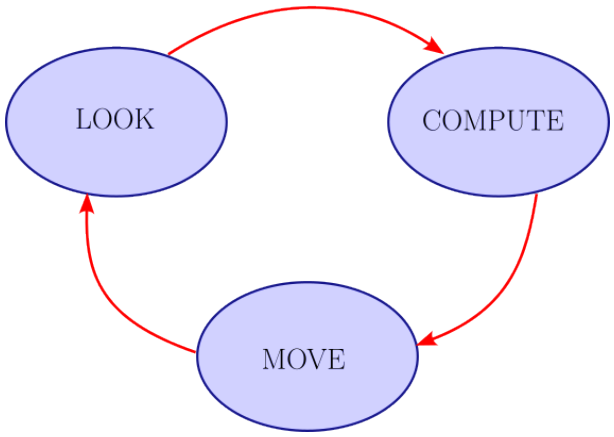


Figure 2.4: Look-compute-move model

Control

Assume that we are able to communicate with the robots from the outside. This external communication could allow us to control a group of robots. This can be done using false or true messages. In our example, we may send a false message so that all robots go up at $k = 2$ (see Figure 2.5).

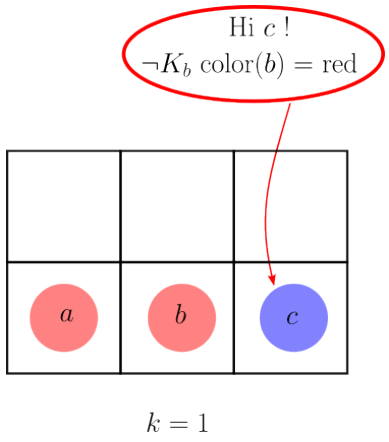


Figure 2.5: With messages (true or false), we can control the behavior of some robots

2.2 Ontic and epistemic states for mobile robots

A mobile robot has a ontic state (position, speed, heading, ...) and an epistemic state (memory). An illustration is provided by Figure 2.6 for one robot.

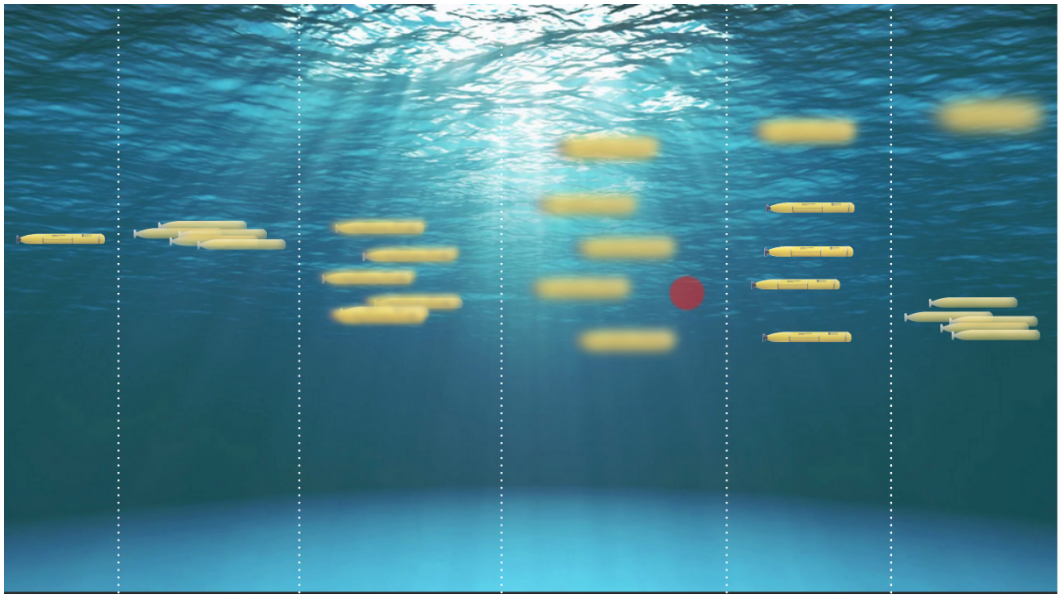


Figure 2.6: The uncertainty on the position is represented by a cloud,
The uncertainty of the knowledge is represented by the blurry appearance

A robot with memory can be represented by the following state equations:

$$\begin{array}{lcll} \dot{\mathbf{x}}(t) & = & \mathbf{f}(\mathbf{x}(t),\mathbf{u}(t)) & \text{(ontic evolution)} \\ \mathbf{y}(t) & = & \mathbf{g}(\mathbf{x}(t)) & \text{(observation)} \\ \boldsymbol{\mu}_{k+1} & = & \boldsymbol{\varphi}(\boldsymbol{\mu}_k,\mathbf{y}(t_k)) & \text{(epistemic evolution)} \\ \mathbf{u}(t_k) & = & \mathbf{h}(\boldsymbol{\mu}_k) & \text{(control)} \end{array}$$

The vector $\boldsymbol{\mu}_k$ corresponds to the memory of the computer and its evolution is discrete in time. Note that we have an hybrid system since the ontic evolution is continuous in time. For simplicity, we consider an analog controller. This simplifies the analysis since all evolutions are now continuous in time. The state equations (see Figure 2.7) become

$$\begin{array}{lcll} \dot{\mathbf{x}}(t) & = & \mathbf{f}(\mathbf{x}(t),\mathbf{u}(t)) & \text{(ontic evolution)} \\ \mathbf{y}(t) & = & \mathbf{g}(\mathbf{x}(t)) & \text{(observation)} \\ \dot{\boldsymbol{\mu}}(t) & = & \boldsymbol{\varphi}(\boldsymbol{\mu}(t),\mathbf{y}(t)) & \text{(epistemic evolution)} \\ \mathbf{u}(t) & = & \mathbf{h}(\boldsymbol{\mu}(t)). & \text{(control)} \end{array}$$

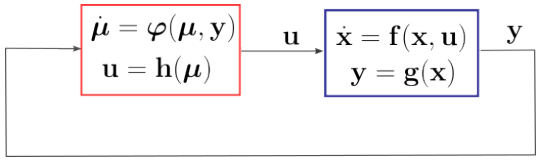


Figure 2.7: Closed loop system with the controller (red) and the actual system (blue)

Which translates into

$$\begin{array}{rcll} \dot{\mathbf{x}}(t) & = & \mathbf{f}(\mathbf{x}(t), \mathbf{h}(\boldsymbol{\mu}(t))) & \text{(ontic evolution)} \\ \dot{\boldsymbol{\mu}}(t) & = & \boldsymbol{\varphi}(\boldsymbol{\mu}(t), \mathbf{g}(\mathbf{x}(t))) & \text{(epistemic evolution)} \end{array}$$

If we define the global state as $\mathbf{z} = (\mathbf{x}, \boldsymbol{\mu})$, we have

$$\dot{\mathbf{z}} = \boldsymbol{\psi}(\mathbf{z})$$

Recall that the state of the robot which is now : $\mathbf{z} = (\mathbf{x}, \boldsymbol{\mu})$, can be decomposed into

- $\mathbf{x}$: the ontic state
- $\boldsymbol{\mu}$: the epistemic state

Some uncertainties are attached to these states.

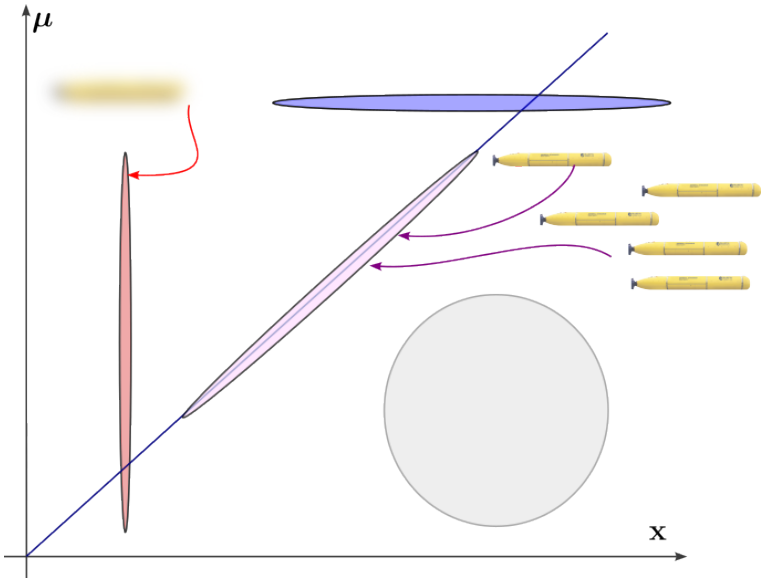


Figure 2.8: A representation of the uncertainties (x-axis: ontic; y-axis: epistemic)

Two types of measurements can be done on the system:

- Perception : We measure $\mathbf{x}$
- Communication : we measure $\boldsymbol{\mu}$

2.3 Attack

An attack can be performed by communicating false information to the robot. Consider for instance a robot which uses a landmark $\mathbf{m}$ for its own localization, as illustrated by Figure 2.9.

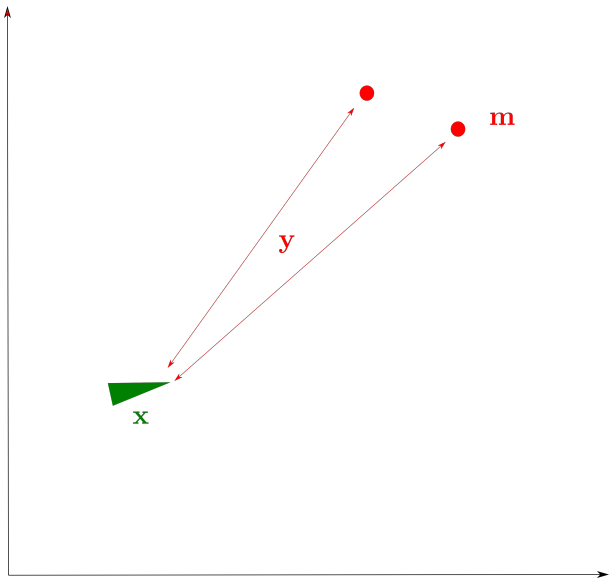


Figure 2.9: A robot uses the landmark $\mathbf{m}$ for its localization

Its measurement may depend on $\mathbf{m}$, that is known by the robot. We thus have an evolution and a measurement equations of the form

$$\begin{array}{rcl} \dot{\mathbf{x}} & = & \mathbf{f}(\mathbf{x},\mathbf{u}) \\ \mathbf{y} & = & \mathbf{g}(\mathbf{x},\mathbf{m}) \end{array}$$

The robot may use $\mathbf{m}$ to estimate its ontic state vector. Denote by $\hat{\mathbf{x}}$ the estimated state vector (see Figure 2.10). The state vector of the robot contains both $\mathbf{x}$ and $\hat{\mathbf{x}}$ which is stored in the computer of the robot.

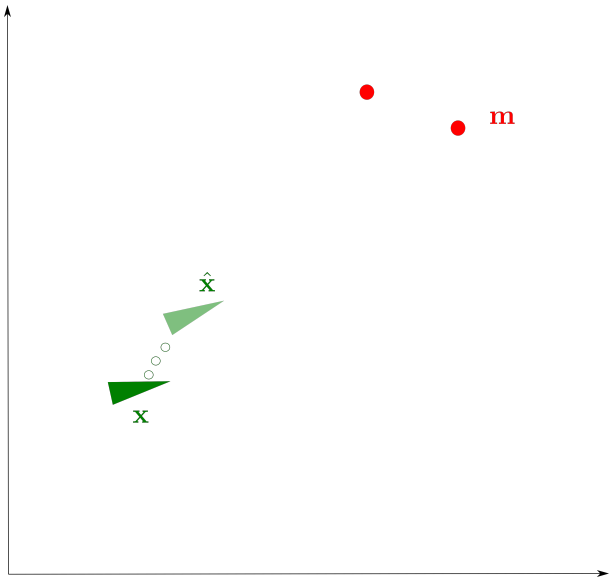


Figure 2.10: The robot estimates its state $\mathbf{x}$ and this estimate $\hat{\mathbf{x}}$ is stored in its memory

This can be done using a Luenberger observer or a Kalman-bucy filter. The state equations of the robot are

$$\begin{array}{rcl} \dot{\mathbf{x}} & = & \mathbf{f}(\mathbf{x},\mathbf{u}) \\ \mathbf{y} & = & \mathbf{g}(\mathbf{x},\mathbf{m}) \\ \dot{\hat{\mathbf{x}}} & = & \hat{\mathbf{f}}(\hat{\mathbf{x}},\mathbf{y},\mathbf{u}) \\ \mathbf{u} & = & \mathbf{r}(\hat{\mathbf{x}}) \end{array}$$

as illustrated by Figure 2.11

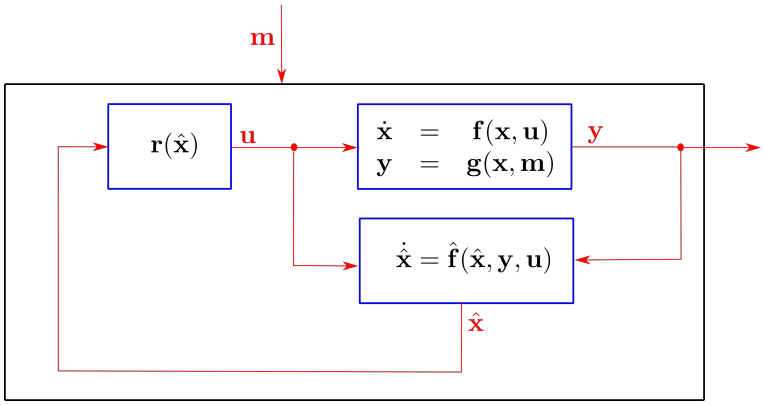


Figure 2.11: Closed loop system. The new input is now $\mathbf{m}$

Equivalently, we have:

$$\begin{array}{rcccl} \dot{\mathbf{x}} & = & \mathbf{f}(\mathbf{x},\mathbf{r}(\hat{\mathbf{x}})) & = & \mathbf{f}(\mathbf{x},\hat{\mathbf{x}}) \\ \dot{\hat{\mathbf{x}}} & = & \hat{\mathbf{f}}(\hat{\mathbf{x}},\mathbf{g}(\mathbf{x},\mathbf{m}),\mathbf{r}(\hat{\mathbf{x}})) & = & \hat{\mathbf{f}}(\mathbf{x},\hat{\mathbf{x}},\mathbf{m}) \end{array}$$

If we set $\mathbf{z} = (\mathbf{x}, \hat{\mathbf{x}})$, we get a state equation of the form

$$\dot{\mathbf{z}} = \boldsymbol{\psi}(\mathbf{z},\mathbf{m})$$

Assume that we can observe the motion of the robot

$$\begin{array}{rcl} \dot{\mathbf{z}} & = & \boldsymbol{\psi}(\mathbf{z},\mathbf{m}) \\ \mathbf{a} & = & \boldsymbol{\eta}(\mathbf{x}) \end{array}$$

where $\mathbf{z} = (\mathbf{x}, \hat{\mathbf{x}})$.

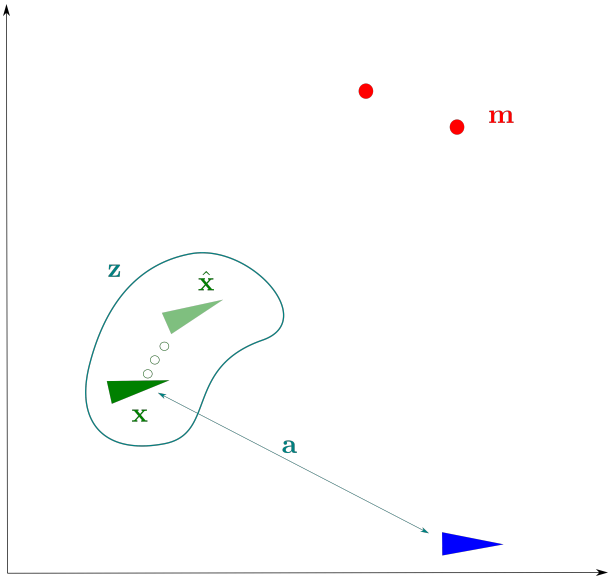


Figure 2.12: The observer (blue) measures a part of the ontic state of the green robot

We can build an observer for $\mathbf{z}$. It can be described by the following state equations:

$$\begin{array}{rcl} \dot{\mathbf{z}} & = & \boldsymbol{\psi}(\mathbf{z}, \mathbf{m}) \\ \mathbf{a} & = & \boldsymbol{\eta}(\mathbf{x}) \\ \dot{\hat{\mathbf{z}}} & = & \hat{\boldsymbol{\psi}}(\mathbf{a}, \hat{\mathbf{z}}) \end{array}$$

where $\hat{\mathbf{z}}$ is an estimation of the state ontic state of the robot and also of its memory. We can now use $\mathbf{m}$ as an input to control $\mathbf{x}$. Recall that the vector $\mathbf{m}$ corresponds to the position of the landmark (see Figure 2.13).

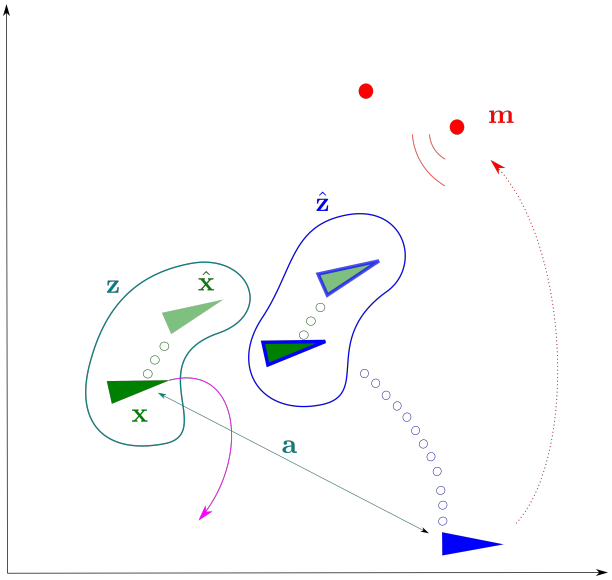


Figure 2.13: The blue robot has to estimate both $\mathbf{x}$ and $\hat{\mathbf{x}}$ from measurements $\mathbf{a}$ of $\mathbf{x}$

For this purpose, we want to build a controller:

$$\mathbf{m} = \boldsymbol{\rho}(\hat{\mathbf{z}}, t)$$

to control both the ontic state and the epistemic state. The closed loop system, with the attacker in the loop is given by Figure 2.14.

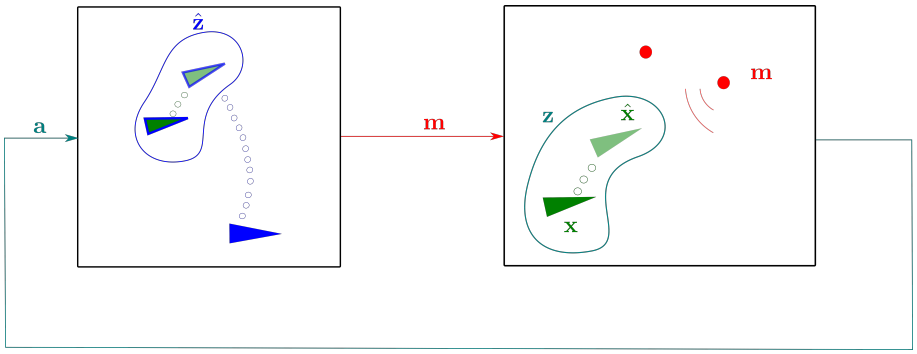


Figure 2.14: Sending false information from $\mathbf{m}$, we can take the control of the robot motion, and also of a part of its memory

EXERCISE 5.– *Cyber attack*

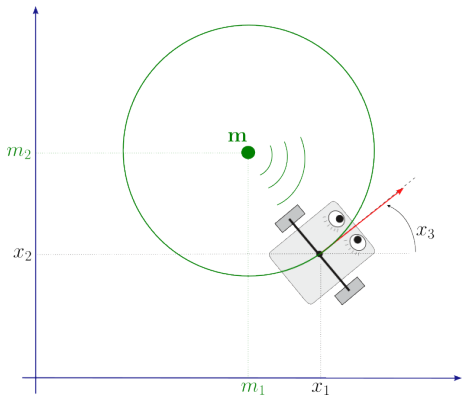
Consider a Dubins car

$$\left\{\begin{array}{lcl} \dot{x}_1 &=& \cos x_3 \\ \dot{x}_2 &=& \sin x_3 \\ \dot{x}_3 &=& u \end{array}\right.$$

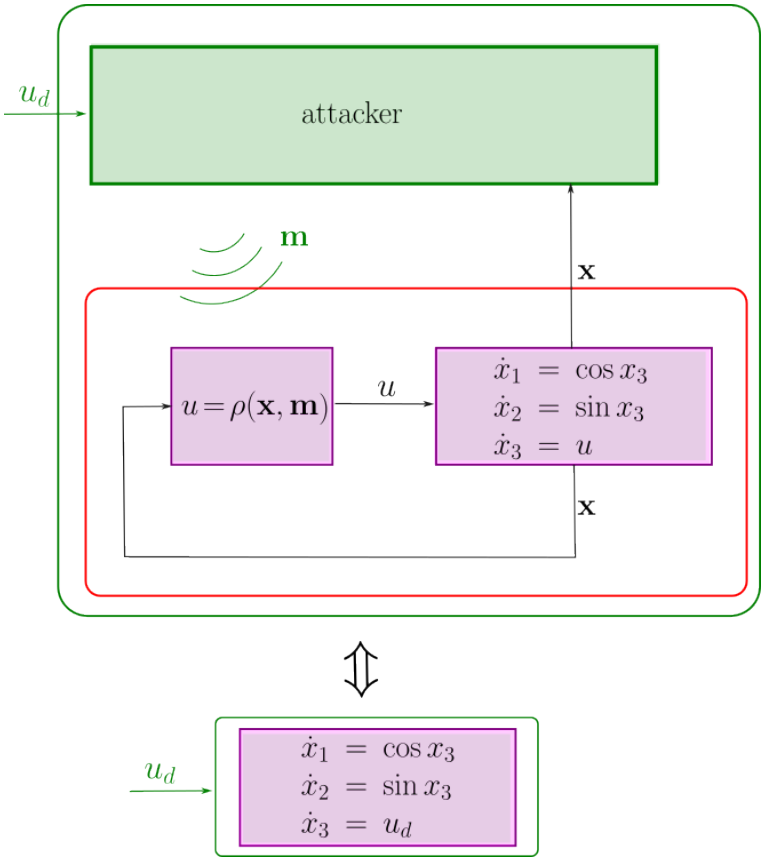
A landmark $\mathbf{m}$ broadcast its position to all world.

1) Propose a controller $u = \rho(\mathbf{x}, \mathbf{m})$ so that the car turns around a landmark $\mathbf{m}$ generating a circle of radius r. To do this use a feedback linearisation method with the output

$$y = \frac{1}{2}\underbrace{(x_1 - m_1)}_{d_1}^2 + \frac{1}{2}\underbrace{(x_2 - m_2)}_{d_2}^2$$



2) Assume that we have hacked the computer of the landmark and that we are able to send fake position for $\mathbf{m}$. Find the fake position to be sent so that we can take the control of the vehicle, as illustrated by the following figure.


u_d
attacker
$\mathbf{m}$
$\mathbf{x}$
$u = \rho(\mathbf{x}, \mathbf{m})$
u
$\dot{x}_1 = \cos x_3$
$\dot{x}_2 = \sin x_3$
$\dot{x}_3 = u$
$\mathbf{x}$
u_d
$\dot{x}_1 = \cos x_3$
$\dot{x}_2 = \sin x_3$
$\dot{x}_3 = u_d$

Chapter 3

Cryptography

Cryptography [5] studies how to secure communications between robots in the presence of adversarial behavior. Practical applications of cryptography include electronic commerce, payment cards, computer passwords, military communications or any communicating robots.

3.1 One-way function

The main algorithms used in cryptography is probably the notion of one-way functions. A one-way function is a function that is easy to compute on every input, but hard to invert given the image of a random input. Such a function is also called a cryptographic hash function. We will now build one-way functions used in cryptography. The conctruction is based on the modular arithmetic.

3.2 Modular arithmetic

This section presents modular arithmetic which relies on classical mathematical notions such as the divisibility of integers, group theory, Bezout equations, ... The resulting algorithm make extensive use of the Euclidian division.

3.2.1 Euclidian division

Given two integers $a \in \mathbb{N}$ and $b \in \mathbb{N}$, with $b \neq 0$, there exist unique integers q and r such that
$a = bq + r$
$0 \leq r < b$,
The integer a is called the *dividend*, b is called the *divisor*, q is called the *quotient* and r is called the *remainder*.
We define the two binary operators % ans // often used in the modular arithmetic

- $a\%b$ which returns the remainder r from a, b
- $a//b$ which returns the quotient q from a, b

For instance, since $17 = 5 \cdot 3 + 2$, we have $17\%5 = 2$ and $17//5 = 3$.

3.2.2 Divisibility

An integer n is *divisible* by a nonzero integer m if there exists an integer k such that $n = km$. We write $m \mid n$

- For all $a \in \mathbb{Z}$, $a \mid a$ that is, divisibility is reflexive.
- If $a \mid b$ and $b \mid c$, then $a \mid c$ that is, divisibility is a transitive relation.
- If $a \mid b$ and $b \mid a$ then $a = b$ or $a = -b$
- If $a \mid b$ and $a \mid c$ then $a \mid (b + c)$.
- $a \mid b \Leftrightarrow ac \mid bc$ for nonzero c .
- For all $a \in \mathbb{Z}$, we have $a \mid 0$.

3.2.3 Greatest common divisor

Given two integers a, b, a common divisor is an integer d such that $d \mid a$ and $d \mid b$. The *greatest common divisor* of a, b is denoted by $gcd(a, b) = a \wedge b$.

For instance

$$12 \wedge 18 = 2 \cdot (6 \wedge 9) = 2 \cdot 3 \cdot (2 \wedge 3) = 6$$

Lemma (Euclide). If a, b are natural numbers then $a \wedge b = (a\%b) \wedge b$.

Proof. Let $r = a\%b$. We have $a = qb + r$ for some integer q. Since $(b \wedge r) \mid b$ and $(b \wedge r) \mid r$, we have $(b \wedge r) \mid qb + r$, *i.e.*, . $(b \wedge r) \mid a$. Therefore $b \wedge r$ is a common divisor of a and b, so that that $b \wedge r | a \wedge b$. On the other hand, since $r = a - qb$ implies that $(a \wedge b) \mid r$. Therefore $(a \wedge b)$ is a common divisor of b and r, so $a \wedge b | b \wedge r$, which forces them to be equal.

This lemma is usually used to compute the $a \wedge b$

Example. For example $100 \wedge 40 = 40 \wedge 20 = 0 \wedge 20 = 20$.

3.2.4 Congruence

Given an integer $m \geq 1$, called a modulus, two integers a and b are said to be congruent modulo m, if there is an integer k such that

$a - b = km$.

Congruence modulo m is a congruence relation, meaning that it is an equivalence relation that is compatible with the operations of addition, subtraction, and multiplication. Congruence modulo m is denoted

$a \equiv b[m]$.

We have the following properties

$$\begin{array}{rcll} a \equiv a[m] & & & \text{(Reflexivity)} \\ a \equiv b[m] & \Rightarrow & b \equiv a[m] & \text{(Symmetry)} \\ a \equiv b[m], b \equiv c[m] & \Rightarrow & a \equiv c[m] & \text{(Transitivity)} \\ a_1 \equiv b_1[m], a_2 \equiv b_2[m] & \Rightarrow & a_1 + a_2 \equiv b_1 + b_2[m] & \\ a_1 \equiv b_1[m], a_2 \equiv b_2[m] & \Rightarrow & a_1 - a_2 \equiv b_1 - b_2[m] & \\ a_1 \equiv b_1[m], a_2 \equiv b_2[m] & \Rightarrow & a_1 \cdot a_2 \equiv b_1 \cdot b_2[m] & \end{array}$$

3.2.5 Diophantine equations

Diophantine equation is an equation with integer coefficients where the solutions are also required to be integers. The simplest examples are the linear ones: given integers a, b, c, find all integers m, n such that $am + bn = c$.

Theorem. Given integers a, b, c, the equation $am + bn = c$ has a solution with $m, n \in \mathbb{Z}$ if and only if $(a \wedge b) \mid c$.

Corollary. Given $a, b \in \mathbb{Z}$ there exist $m, n \in \mathbb{Z}$ such that $am + bn = a \wedge b$.

Extended Euclide algorithm. The extended Euclidean algorithm solves the equation

$$\left\{ \begin{array}{c} ax + by = z \\ z = a \wedge b \\ a, b, x, y, z \in \mathbb{Z} \end{array} \right.$$

where a, b are known.

The extended Euclide algorithm can be written as a function $E(a, b)$ which provides the solution (x, y, z) of the problem.

Proposition 1. *The function E can be defined by induction as*

$$E \begin{pmatrix} a \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ a \end{pmatrix}$$

and

$$E \begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} y \\ x - (a//b) * y \\ z \end{pmatrix} \text{ where } \begin{pmatrix} x \\ y \\ z \end{pmatrix} = E \begin{pmatrix} b \\ a\%b \end{pmatrix}$$

Proof. Since in the case where $b = 0$, we have $z = a \wedge b = a$ and $x = 1$,we get

$$E(a, 0) = (1, 0, a)$$

Moreover, we have, if $z = a \wedge b$, we have

$$\begin{array}{rl} & ax + by = z \\ \Leftrightarrow & (bq + r)x + by = z \quad \text{where} \quad q = a//b, r = a\%b \\ \Leftrightarrow & b(qx + y) + rx = z \\ \Leftrightarrow & \left\{ \begin{array}{c} by' + rx = z \\ y' = qx + y \end{array} \right. \end{array}$$

The equation $by' + rx = z$ is simpler to solve than $ax + by = z$. □

The corresponding Python code is:

```
def E(a,b): # extended Euclide algorithm
   if b==0:  return 1,0,a
   else:
      x,y,z=E(b,a%b)
      return y,x-(a//b)*y,z
```

The greatest common divisor for $a \wedge b$ is obtained by

```
def gcd(a,b):
   _,_,z=E(a,b)
   return z
```

3.2.6 Group $\left(\left(\frac{\mathbb{Z}}{p\mathbb{Z}}\right)^*, \cdot\right)$

Consider the group $\left(\left(\frac{\mathbb{Z}}{p\mathbb{Z}}\right)^*, \cdot\right)$ where p is prime. The star operator means that 0 has been removed. The operator of the group is the multiplication. For instance, if we consider the group $\left(\frac{\mathbb{Z}}{5\mathbb{Z}}\right)^* = \{1, 2, 3, 4\}$. The multiplication table is

$\cdot$	1	2	3	4
1	1	2	3	4
2	2	4	1	3
3	3	1	4	2
4	4	3	2	1

We have $a \cdot b = (a * b)\%5$, where $*$ is the classical multiplication in $\mathbb{N}$. For instance, $3 \cdot 4 = 12\%5 = (5 * 2 + 2)\%5 = 2$.

The modular inverse a^{-1} of a is the element which satisfies $a^{-1} \cdot a = 1$. For instance $3^{-1} = 2$, since $2 \cdot 3 = 1$.

Ferma's little theorem. For any prime number p and any integer a, one has

$$a^{p-1} = 1[n]$$

For instance $3^4 = 1$ in $\left(\frac{\mathbb{Z}}{5\mathbb{Z}}\right)^*$. This theorem can help to the inverse of a number, when p is huge. For instance, the inverse of 3 is $3^3 = 2$. Indeed $3 \cdot 3^3 = 1$. In cryptography, we always have to keep in mind that inversion has to be difficult if we want to build one-way functions.

3.2.7 Monoid $\left(\frac{\mathbb{Z}}{n\mathbb{Z}}, \cdot\right)$

When n is not prime, the set $\left(\frac{\mathbb{Z}}{n\mathbb{Z}}, \cdot\right)$ is not a group with respect to the multiplication. It is a monoid only. For instance $\frac{\mathbb{Z}}{4\mathbb{Z}} = \{0, 1, 2, 3\}$. The multiplication table is

$\cdot$	0	1	2	3
0	0	0	0	0
1	0	1	2	3
2	0	2	0	2
3	0	3	2	1

Some elements have an inverse (here 1,3), some not. Only elements that are prime to n have an inverse.

3.2.8 Modular inverse

In $\left(\frac{\mathbb{Z}}{n\mathbb{Z}}\right)$, the modular inverse of a is x if $a \cdot x = 1$. The inverse exists if $a \wedge n = 1$. Equivalently, we have $a * x \equiv 1[n]$. To find the inverse, we have to solve the Bezout equation $a * x + n * y = 1$. This can be done using the Euclide algorithm.

Take for instance $\left(\frac{\mathbb{Z}}{4\mathbb{Z}}, \cdot\right)$ and $a = 3$. To find an inverse x of a, we first check that $a \wedge n = 1$. We have to solve $a * x + n * y = 1$, i.e., $3 * x + 4 * y = 1$. We get $x = -1$ and $y = 1$. Now, $x\%4 = 3$. We get $3^{-1} = 3$ in $\frac{\mathbb{Z}}{4\mathbb{Z}}$.

The Python function for the mudular inverse is given by

```
def modular_inverse(a,n):
   x,_,_=E(a,n)
   return x%n
```

3.2.9 Discrete exponential

An example of one-way function is the discrete exponential (or *modular exponentiation)*. Modular exponentiation can be done in polynomial time. Inverting this function requires computing the discrete logarithm. This will be done is the group $\left(\frac{\mathbb{Z}}{p\mathbb{Z}}\right)^*$, where p is a prime number.

In the group in $\left(\frac{\mathbb{Z}}{5\mathbb{Z}}\right)^*$, $3^4 = 1$ because $3 * 3 * 3 * 3 = 81$ and $81 \equiv 1[5]$. We have

$$\begin{aligned} 3^0 &= 1 \\ 3^1 &= 3 \\ 3^2 &= 4 \\ 3^3 &= 2 \\ 3^4 &= 1 \\ 3^5 &= 3 \\ 3^6 &= 4 \\ 3^7 &= 2 \end{aligned}$$

Squaring method for the exponentiation

Assume that we have to compute 2^{71} in $\left(\frac{\mathbb{Z}}{5\mathbb{Z}}\right)^*$. Of course, we do not want to perform 71 multiplications. We have

$$\begin{aligned}
3^{71} &= 3\cdot 3^{70}\\
&= 3\cdot 3^{35}\cdot 3^{35}\\
&= 3\cdot (3^{35})^2\\
&= 3\cdot (3\cdot 3^{34})^2\\
&= 3\cdot \left(3\cdot (3^{17})^2\right)^2\\
&= 3\cdot \left(3\cdot (3\cdot 3^{16})^2\right)^2\\
&= 3\cdot \left(3\cdot \left(3\cdot (3^{8})^2\right)^2\right)^2\\
&= 3\cdot \left(3\cdot \left(3\cdot \left((3^{4})^2\right)^2\right)^2\right)^2\\
&= 3\cdot \left(3\cdot \left(3\cdot \left(\left((3^{2})^2\right)^2\right)^2\right)\right)^2\\
&= 3\cdot \left(3\cdot \left(3\cdot \left(\left((3^{2})^2\right)^2\right)^2\right)\right)^2\\
&= 3\cdot \left(3\cdot \left(3\cdot \left((4^{2})^2\right)^2\right)^2\right)^2\\
&= 3\cdot \left(3\cdot \left(3\cdot (1^{2})^2\right)^2\right)^2\\
&= 3\cdot (3\cdot 3^2)^2\\
&= 3\cdot (3\cdot 4)^2\\
&= 3\cdot 2^2\\
&= 3\cdot 4 = 2
\end{aligned}$$

We have used here a squaring method where the idea is to square the expression as much as possible. The squaring makes the exponentiation fast. More precisely, if we have to compute a^m, where $a \in \left(\frac{\mathbb{Z}}{p\mathbb{Z}}\right)^*$ and $m \in \mathbb{N}$ is usually a huge number, we can get an algorithm in $\log m$, or equivalently which is linear with respect the the length of m. In out example, we has $m = 71$ which means that the length of m is equal to 2.

Euclidian division for the exponentiation

Assume again that we have to compute 2^{71} in $\left(\frac{\mathbb{Z}}{5\mathbb{Z}}\right)^*$. We know that $3^4 = 1$. Moreover $71 = 17\cdot 4+3$. Thus

$$2^{71} = 3^{17\cdot 4+3} = \left(3^4\right)^3 3^3 = (1)^3 3^3 = 2.$$

Discrete Logarithm

Let G be a finite abelian group of cardinality n. Denote its group operation by multiplication. Consider a primitive element $\alpha \in G$ and another element $\beta \in G$. The discrete logarithm problem is to find the smallest positive integer k, such that:

$$\alpha^k = \beta$$

The integer k is termed the discrete logarithm (or modular logarithm) of β to the base α. We write $k = \log_\alpha \beta$.

Consider again the group $\left(\frac{\mathbb{Z}}{5\mathbb{Z}}\right)^*$. Since

$$\begin{aligned} 3^1 &= 3 \\ 3^2 &= 4 \\ 3^3 &= 2 \\ 3^4 &= 1 \end{aligned}$$

we have

$$\begin{aligned} \log_3(1) &= 4 \\ \log_3(2) &= 3 \\ \log_3(3) &= 1 \\ \log_3(4) &= 2 \end{aligned}$$

There is no algorithm to compute efficiently the discrete logarithm.

3.3 Asymmetric cryptography

Alice wants Bob sends her a short message m, securely. Here m is assumed to be an integer (not too large). If the message to be sent is large (a file, for instance), the message has to be decomposed into symbols (letters for instance) and these symbols have to be transformed into integers. We assume here that m is a small integer (for instance in $\{1, \ldots, 26\}$ if we use the letters of the Alphabet). Alice creates a pair of two functions H, H^{-1} such that $H \circ H^{-1} = Id$, where H is a one-way function, called a *cryptographic hatching function*. She sends H to Bob and everybody can capture H which should now be considered as public. Bob generates $c = H(m)$ which corresponds to the coded message. Bob sends c to Alice and c becomes public. Alice receives c and computes $m = H^{-1}(c)$. Only Alice can do this, since only her knows H^{-1}. At this step, both Alice and Bob know m and nobody else. The process is illustrated by Figure 3.1. If Bob wants to send another message m there is no need to build another pair (H, H^{-1}).

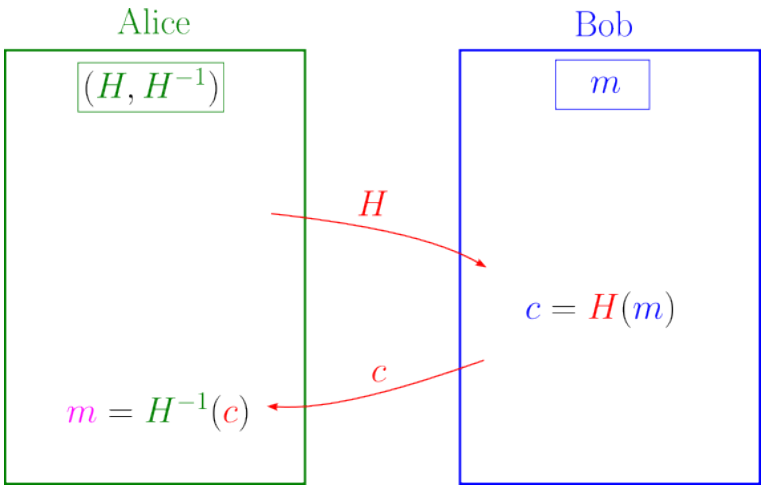


Figure 3.1: Red: Public; Green: Known by Alice only;
Blue: known by Bob only; Magenta: known by Bob and Alice only

Long message. If Bob has a long message M to send, he has to decompose M into many integers: $m_1, m_2, \ldots$ and send the $c_i = H(m_i)$. A unique function H is needed.

Naive approach. We propose here a very naive approach which gives the principle of the RSA coding explained just after. Alice chooses an number e for which only she knows the inverse d. The numbers e, m are assumed to be in some multiplicative group where the exponent and the logarithm exist. Then Bob computes $c = m^e$ and sends c to Alice. Alice computes c^d to get m. Indeed, we have

$$c^d = (m^e)^d = m^{ed} = m^1 = m.$$

Equivalently, Alice could have done the following computation

$$m = \exp\left(d \cdot \log c\right).$$

Now, the notion of inverse, log and exp cannot be used in a modular context as it is done in $\mathbb{R}$. Moreover, we have to keep in mind that to get a one-way function, the inverse of e should be difficult to compute. It is the case in a modular arithmetic, *i.e.*, in $\left(\frac{\mathbb{Z}}{n\mathbb{Z}}\right)^*$ where n is large. Moreover, in $\left(\frac{\mathbb{Z}}{n\mathbb{Z}}\right)^*$, the computation are cyclic which creates some ambiguities to be avoided. And if m is not a prime number, a number e may have no inverse.

The naive approach inspires the RSA cryptosystem that is now explained.

3.3.1 RSA cryptosystem

RSA is a public-key cryptosystem is one of the oldest widely used for secure data transmission. The name "RSA" comes from the names of R. Rivest, A. Shamir and L. Adleman, who described first the algorithm in 1977. With RSA, the one-way function H is parametrized by the pair (p, q), where p, q are two prime numbers (see Figure 3.2). The functions H and H^{-1} are obtained using the following procedure:

- $n = p \cdot q$. The two numbers should be large enough to have $m < p \cdot q$.
- $\varphi_1 = (p-1) \cdot (q-1)$. Note that $\varphi_1 = \varphi(n) = \varphi(p \cdot q) = \varphi(p \cdot q) = (p-1) \cdot (q-1)$ where φ corresponds to the Euler's totient function, *i.e.*, $\varphi(n)$ corresponds to the number of elements smaller that n that are relative prime to n.
- e is any integer > 1 such that $e \wedge \varphi_1 = 1$.
- d is the modular inverse of e in $\left(\frac{\mathbb{Z}}{\varphi_1 \mathbb{Z}}\right)$. It satisfies $de = 1[\varphi_1]$.
- Return $\begin{pmatrix} H : m & \mapsto & m^e \,\%\, n \\ H^{-1} : c & \mapsto & c^d \,\%\, n \end{pmatrix}$

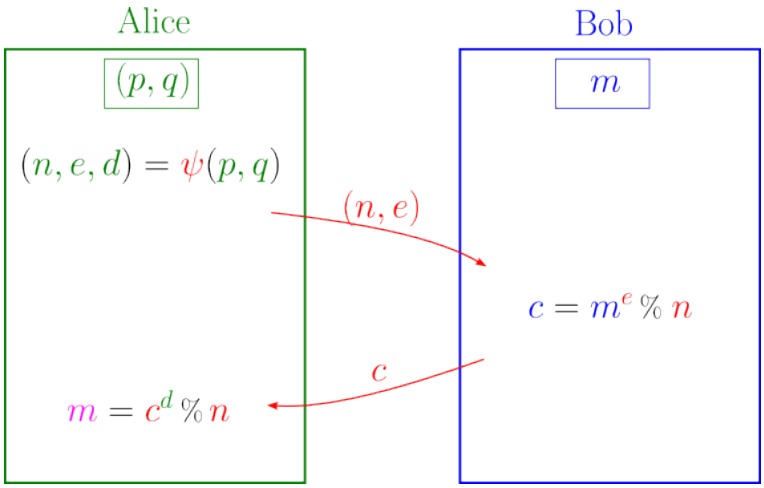


Figure 3.2: (n, e) are public; (p, q, d) are private
The message m of bob is coded into c and then decoded by Alice
Many messages can be sent using the same keys

The fact that Alice has decoded the message, *i.e.*, the fact that $m = c^d \,\%\, n$, comes from the following theorem:

Theorem 2. *If $n = p \cdot q$ where p, q are prime. If $d, e \in \mathbb{N}$ are such that $de \equiv 1[(p-1)(q-1)]$, and if $m \in \mathbb{N}$ then*

$$(m^d)^e \equiv m[n].$$

If $m < n$, we get $(m^d)^e \equiv m$.

Remark 3. The integer d is called the *trapdoor*. It corresponds to the secret information needed to invert the one-way function H.

Example 4. For $p = 3, q = 11$,we have $n = 33$, $\varphi_1 = (3-1)\cdot(11-1) = 20$, $e = 3$, and $d = 7$ (indeed $7 * 3 \equiv 1[20]$). For instance, if $m = 16$, we have

$$c = H(m) = m^e \,\%\, n = 16^3 \,\%\, 33 = 4$$

and

$$H^{-1}(c) = c^d \,\%\, n = 4^7 \,\%\, 33 = 16$$

The following example is similar, with larger number

Example 5. For $p = 23, q = 61$,we have $n = 23 * 61 = 1403$, $\varphi_1 = (23-1)\cdot(61-1) = 1320$, We have to take e such that $e \wedge \varphi_1 = 1$. Take for instance $e = 7$. We have to find an inverse d in $\left(\frac{\mathbb{Z}}{\varphi_1\mathbb{Z}}\right)$, *i.e.*, an integer e such that $ed \equiv 1[\varphi_1]$. Since $e \wedge \varphi_1 = 1$ the inverse of e exists in $\left(\frac{\mathbb{Z}}{\varphi_1\mathbb{Z}}\right)$. From Bezout theorem, there exists $d, f \in \mathbb{N}$ such that $de + f\varphi_1 = 1$. The pair (d, f) can be obtained using the Euclide algorithm. Thus $de \equiv 1[\varphi_1]$. We can take $d = 943$.

For instance, if $m = 345$, we have

$$c = H(m) = m^e \,\%\, n = 345^7 \,\%\, 1403 = 1058$$

and

$$H^{-1}(c) = c^d \,\%\, n = 1058^{943} \,\%\, 1403 = 345$$

The Python program corresponding to RSA is now given.

```
def generate_keys(p,q):
   n=p*q
   φ1=(p-1)*(q-1)
   e=random.randrange(1,φ1)
   while gcd(e,φ1)!=1: e=random.randrange(1,φ1)
   d=modular_inverse(e,phi1)
   return n,e,d
def encrypt(e,n,M):
   C=[pow(ord(m),e,n) for m in M]
   return C
def decrypt(d,n,C):
      M=[chr(pow(c,d,n)) for c in C]
      return M
p,q = ... # Two large prime numbers
n,e,d=generate_keys(p,q)
print("Public Key: e = ",e)
print("Private Key: d = ",d)
```

```
M=['H','e','l','l','o']
print("Original Message: ",M)
C=encrypt(e,n,M)
print("Encrypted Message: C = ",C)
M=decrypt(d,n,C)
print("Decrypted Message:",M)
```

Chapter 4

Leader election

4.1 Principle

We consider a group of robots. Each robot is able communicate with one, two or more robots of the group. Each robot has its own memory that is not shared with the others. To be able to work all togather, the robots have to choose a leader, *i.e.*, a single robot to take the decision for all. In distributed robotics, leader election [6] is a fundamental task that has to be done in a distributed manner.

At the initialization, all robots do not know which robot is the leader. They do not know how many robots exist in the group. After a leader election algorithm has been run, each robot throughout the network recognizes a particular, unique robot as the leader. To elect a leader, the robots have to break the symmetry among them. This can be done for instance by choosing randomly a name or equivalently an identity number (or address). The communication graph is made with nodes, one for each robot (see Figure 4.1). An arc $a \rightarrow b$ of this graph means that Robot a can communicate with Robot b.

We assume that

- The communication graph is symmetric, *i.e.*, if $a \rightarrow b$ then $b \rightarrow a$.
- The graph is finite and may contain cycles.
- The graph is connected.
- The robots are asynchronous, *i.e.*, they have different clocks and they compute at arbitrary speeds.

.

A valid leader election algorithm must meet the following conditions:

- Termination: the election finishes in a finite time with a high probability.

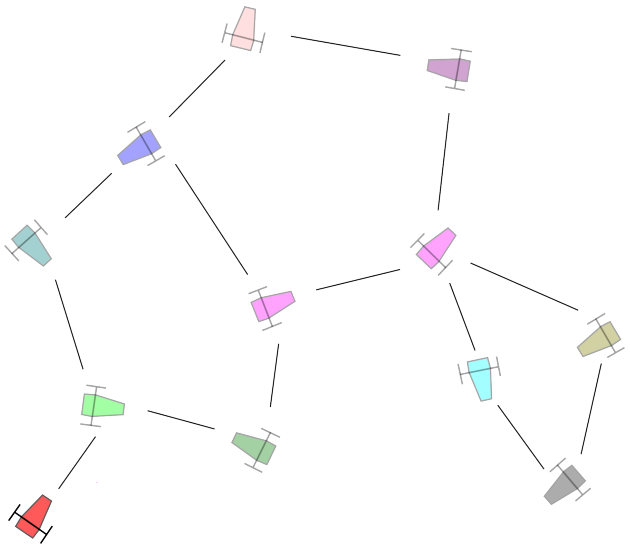


Figure 4.1: 12 robots have to take a common decision.
An arc between two robots means that they can communicate directly

- Uniqueness: there is exactly one robot that considers itself as leader.

- Agreement: all other robots know who the leader is.

4.2 Election

The first step of the election is that each robot chooses a random number in $\{1, \ldots, i_{\max}\}$ called the address. If $i_{\max}$ is large enough, we can assume that all robots have a different address (see Figure 4.2). For simplicity, we take here addresses that are small integers. In the real world, the addresses are not and not given under the form of an integer as for instance:

1. IPv4: 32-bit addresses like 192.168.0.1.

2. IPv6: 128-bit addresses like 2001:0db8:85a3:0000:0000:8a2e:0370:7334.

To select the leader, a possibility is to take the node with the largest address. At this step, no communication has been done yet, nobody knows who is the leader. The set of all addresses is denoted by $I = \{i_1, i_2, \ldots\}$.

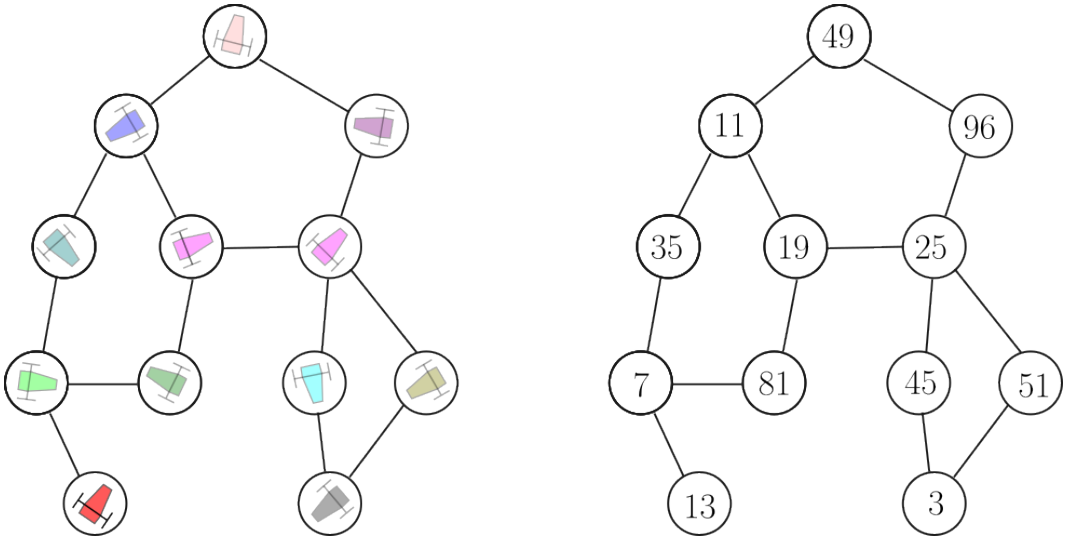


Figure 4.2: The robots choose randomly an address.The red robot chooses the number 13

The leader election can be done using the following procedure (see Figure 4.3).

- Each robot $R_i, i \in I$ has in its own memory a variable, *local leader* ℓ_i, corresponding to the largest address they know. At the initialization, $\ell_i = i$.

- Each robot $R_i, i \in I$ takes the updates its local leader, considering all its neighbors. Equivalently,

$$\ell_i = \max_{k \in \{i\} \cup \{j \,|\, i \to j\}} \ell_j.$$

- After a fixed time T, long enough, all robots are assumed to have the same ℓ_i, *i.e.*, we have $\ell_i = \max I$. The leader election is done.

In the figure below, we have $I = \{3, 7, 11, 13, 19, 25, 35, 45, 49, 51, 81, 96\}$. The leader has the address $i = 96$.

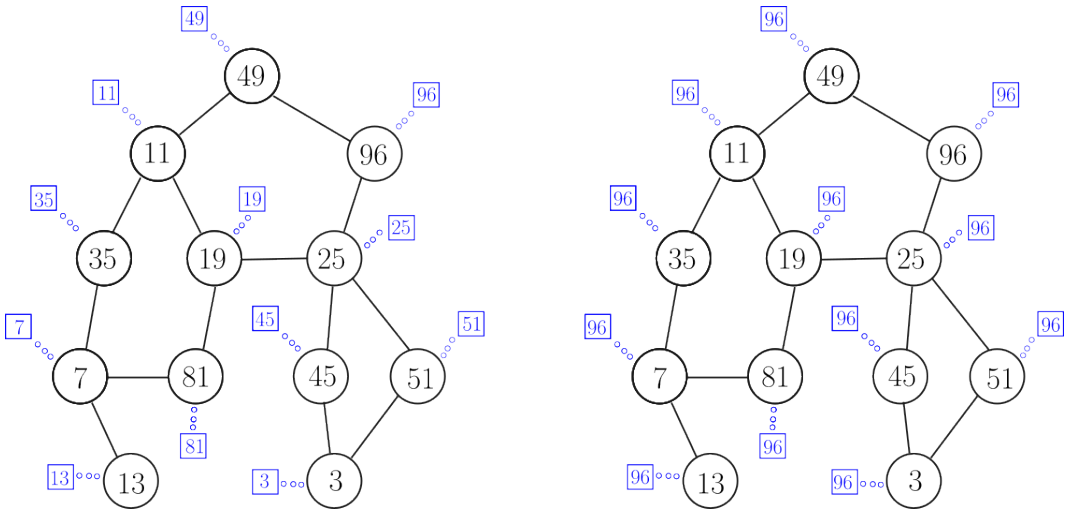


Figure 4.3: Left: Initialization: $\ell_i = i$;Right: the leader election is done: $\ell_i = \max I = 96$

4.3 Span tree

A span tree of the graph G is a tree which contains all nodes of G. In this section, we show how a span tree can now be built. We take the leader as the root of the tree. Each node memorizes its father. To each node i, is also attached an integer d_i which corresponds to distance to the root. To build the span tree, we follow the following procedure.

- **Step 0**. For each $i \in I$, $d_i = \infty$.
- **Step 1**. $d_{\max I} = 0$, *i.e.*, the distance from the leader to itself is 0.
- **Step 2**. For each $i \in I$, such that $d_i = \infty$, if $\exists k \in \{j \,|\, i \to j\}$, with $d_k < \infty$, set $d_i = d_k + 1$ and father$(i) = k$

After a fixed time T, long enough, we assume that $\forall i \in I$, $d_i < \infty$. We now have a span tree with the relation $i \hookrightarrow j$ if $j =$ father(i). The span tree is represented by the red arc of Figure 4.4. Each node memorizes the distance to the root and its father with respect to $\hookrightarrow$. Since $d_{96} = 0$, the robot 96 has no father. The leaves of the tree are $13, 81, 45, 3$. Node 25 has the father 96 and 3 sons: $19, 45, 51$. Once the span tree is built, all communications can be limited to the arcs of the span tree. Since there exists a unique communication path to go from node i to node j, the collision between messages can be avoided. Of course, shortcuts such as $7 \to 81$ would be more efficient than taking $7 \hookrightarrow 35 \hookrightarrow 11 \hookrightarrow 49 \hookrightarrow 96 \hookrightarrow 25 \hookrightarrow 19 \hookrightarrow 81$ but this may create ambiguities.

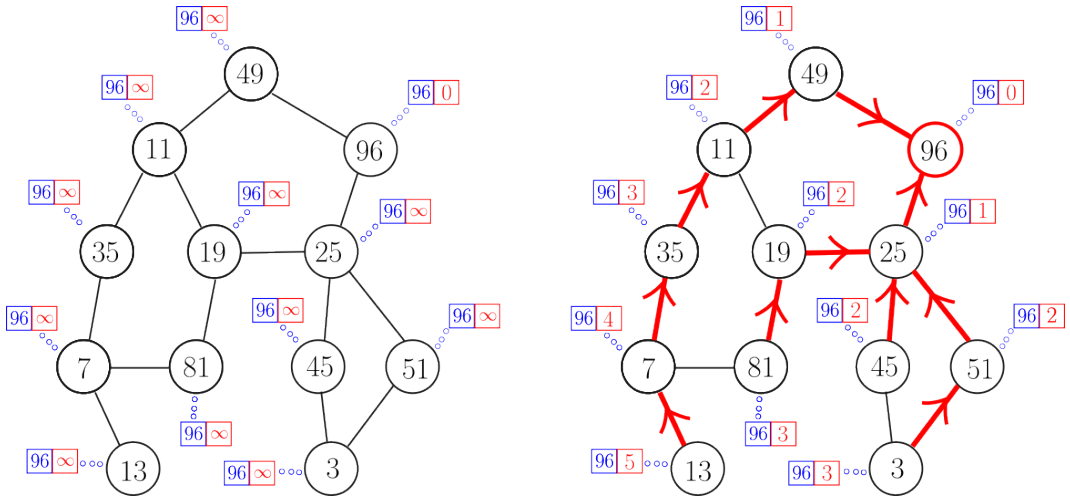


Figure 4.4: Left: Initialization: $d_i = \infty$ except for the leader, $d_{96} = 0$;
Right: span tree is constructed.
The red arc corresponds to the relation $\hookrightarrow$ and points toward the father

Number of span trees. We have proposed a distributed manner to get one span tree. For a given communication graph G, the number N of span trees is usually huge. If G is already a tree, we have $N = 1$. If N is small, a single cut in G can make it disconnected. It is important to get an idea of the value N to have an idea of the robustness of the connection graph. Using the Kirchhoff theorem, we can easily compute N. For this, we need to compute the Laplacian matrix of G. The Laplacian matrix is given in Figure 4.5:

$$\mathbf{L} = \begin{array}{c} \begin{array}{cccccccccccc} 3 & 7 & 11 & 13 & 19 & 25 & 35 & 45 & 49 & 51 & 81 & 96 \\ \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \end{array} \\ \left(\begin{array}{cccccccccccc} 2 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & -1 & 0 & 0 \\ 0 & 3 & 0 & -1 & 0 & 0 & -1 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 3 & 0 & -1 & 0 & -1 & 0 & -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 3 & -1 & 0 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & -1 & 4 & 0 & -1 & 0 & -1 & 0 & -1 \\ 0 & -1 & -1 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 & 0 & -1 & 0 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 & -1 \\ -1 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 2 & 0 & 0 \\ 0 & -1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & -1 & 0 & 0 & 2 \end{array} \right) \end{array} \begin{array}{l} \\ \\ \leftarrow 3 \\ \leftarrow 7 \\ \leftarrow 11 \\ \leftarrow 13 \\ \leftarrow 19 \\ \leftarrow 25 \\ \leftarrow 35 \\ \leftarrow 45 \\ \leftarrow 49 \\ \leftarrow 51 \\ \leftarrow 81 \\ 96 \end{array}$$

Figure 4.5: Laplacian matrix of the communication graph

The matrix is square $n \times n$, where $n = 12$ is the number of nodes. The diagonal (blue) corresponds to the number of neighbors of each node. The entries -1 indicates a neighborhood relation. The sum for each line and each column is zero. To get the number of span trees we compute one cofactor of $\mathbf{L}$. Note that all the cofactors of $\mathbf{L}$ are identical: this is a property of a Laplacian matrix. To compute one cofactor, we remove one line and one column to get a square matrix $\mathbf{L}^*$ with $n-1$ lines and $n-1$ columns. The determinant of $\mathbf{L}^*$ corresponds to the number N of span trees in G. For our example, we get $N = 96$, which is small compared to the number $N_c = 12^{12-2} = 61917364224$ (we used here the Cayley's formula: n^{n-2}) that would have obtained for a complete graph G (*i.e.*, G has an arc between any pair of nodes). The ratio $\frac{N}{N_c} = 2 \cdot 10^{-9}$, which a measure of the graph connectivity, is small. It means that connection graph may loose its connectivity if one or few cuts are performed in the connection graph.

4.4 Counting

Using the span tree, we are now able to count the number of robots. This procedure guarantees that the leader election has succeeded and that the span tree has been built. We define

$$\begin{array}{rcl} \text{father}(i) & = & \{j \,|\, i \hookrightarrow j\} \\ \text{sons}(i) & = & \{j \,|\, j \hookrightarrow i\} \end{array}$$

For a given i, father(i) is either a singleton or empty. In our example, father$(11) = \{49\}$ and father$(96) = \emptyset$. For a given i, sons(i) may contain several nodes. For instance sons$(25) = \{19, 45, 52\}$. A set with no son is called a leaf. For instance, son$(81) = \emptyset$. Thus 81 is a leaf of the span tree.

The principle of the counting is that each node counts the descendants, as illustrated by Figure 4.6. We get the following procedure.

- **Step 0**. For any leaf i of the span tree, set $\eta_i = 1$. Otherwise $\eta_i = 0$.
- **Step 1**. For each $i \in I$, if $\forall j \in son(i), \eta_j > 0$,set $\eta_i = 1 + \sum_{j \in son(i)} \eta_j$. This means that if a node i has all its sons (elements of $son(i)$) that had already made the counting then, by summing $\eta_j, j \in son(i)$, it knows how many descendants it has. It has also to count itself.
- **Step 2**. Repeat Step 1 until $\eta_{\max I} \neq 0$. The integer $\eta_{\max I}$ corresponds to the number of nodes.

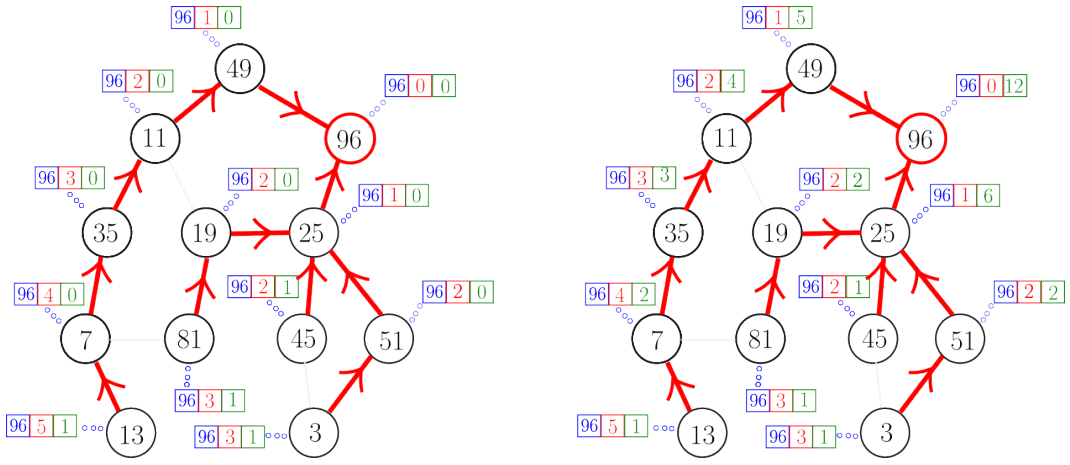


Figure 4.6: Left: Initialization: for all leafs, $\eta_i = 1$;
Right: Each node has counted its descendants in the span tree plus itself

4.5 Communicating

Once a leader election has been done and a span tree has been built, a communication between robots in of the group can be done easily.

Broadcasting

If the leader wants to send a message to all. The strategy is that each robot has to communicate the message received from the father to all sons.

Blackboard

A common knowledge can be set up. It corresponds to some kind of blackboard that anybody can see in the group. The leader owns the blackboard and broadcasts its content as soon as a modification is done.

4.6 Point to point

We want a communication system where that Robot a sends a message to Robot b. Each node has to know the address of all its descendants. We define the function

$$\text{desc}(i) = \text{sons}(i) \cup \bigcup_{j \in \text{sons}(i)} \text{desc}(i)$$

For instance $\text{desc}(25) = \{81, 19, 45, 51, 3\}$. The construction of this function in the graph can be done using a propagation from the leaves to the root, as for the counting. To communicate a message m from node a to node b. We apply the following rules:

- **Rule 1**. If node i received m and if $i = b$, do nothing.
- **Rule 2**. If node i received m and if $b \in \text{desc}(i)$, then i sends m to all $j \in \text{sons}(i)$.
- **Rule 3**. If node i received m and if $b \notin \text{desc}(i)$, then i sends m to $\text{father}(i)$.

Chapter 5

Swarms

Swarm robotics is the study of how to design independent systems of robots without centralized control. The emerging swarming behavior of robotic swarms is created through the interactions between individual robots and the environment. This idea emerged on the field of artificial swarm intelligence, as well as the studies of insects, ants and other fields in nature, where swarm behavior occurs.

Relatively simple individual rules can produce a large set of complex swarm behaviors. A key component is the communication between the members of the group that build a system of constant feedback. The swarm behavior involves constant change of individuals in cooperation with others, as well as the behavior of the whole group.

EXERCISE 6.– *Group of robots*

See the correction video at https://youtu.be/1Of1htovXp4

Consider a group of $m = 20$ carts the motion of which is described by the state equation

$$\begin{cases} \dot{x}_1 &= x_4 \cos x_3 \\ \dot{x}_2 &= x_4 \sin x_3 \\ \dot{x}_3 &= u_1 \\ \dot{x}_4 &= u_2 \end{cases}$$

where (x_1, x_2) corresponds to the position of the cart, x_3 to its heading and x_4 to its speed.

1) Provide a controller for each of these robots so that the ith robot follows the trajectory

$$\begin{pmatrix} \cos(at + \frac{2i\pi}{m}) \\ \sin(at + \frac{2i\pi}{m}) \end{pmatrix}$$

where $a = 0.1$. As a consequence, after the initialization step, all robots are uniformly distributed on the unit circle, turning around the origin.

2) By using a linear transformation of the unit circle, change the controllers for the robots so that all robots stay on a moving ellipse with the first axis of length $20 + 15 \cdot \sin(at)$ and the second axis of length 20. Moreover, we make the ellipse rotating by choosing an angle for the first axis of $\theta = at$. Illustrate the behavior of the controlled group.

EXERCISE 7.– *Convoy*

See the correction video at https://youtu.be/OdRBFO_51s0

Let us consider one robot $\mathcal{R}_A$ described by the following state equations:

$$\begin{cases} \dot{x}_a &= v_a \cos \theta_a \\ \dot{y}_a &= v_a \sin \theta_a \\ \dot{\theta}_a &= u_{a1} \\ \dot{v}_a &= u_{a2} \end{cases}$$

where v_a is the speed of $\mathcal{R}_A$ the robot, θ_a its orientation and (x_a, y_a) the coordinates of its center.

1) As for Exercise **??**, propose a controller for $\mathcal{R}_A$ to follow the trajectory:

$$\begin{cases} \hat{x}_a(t) &= L_x \sin(\omega t) \\ \hat{y}_a(t) &= L_y \cos(\omega t) \end{cases}$$

with $\omega = 0.1$, $L_x = 20$ and $L_y = 5$. Illustrate the behavior of the control with a sampling time $dt = 0.03\,\text{sec}$.

2) We want that $m = 6$ other robots with the same state equations follow this robot taking exactly the same path. The distance between two robots should be $d = 5m$. To achieve this goal, we

propose to save every $ds = 0.1m$ the value of the state of $\mathcal{R}_A$ and to communicate this information to the m followers, to synchronize the time with the traveled distance. For this, we propose to add a new state variable s to $\mathcal{R}_A$ which corresponds to the curvilinear value that could have been measured by a virtual odometer. Each time the distance ds has been measured by the virtual odometer, s is initialized to zero and the value for the state of $\mathcal{R}_A$ is broadcast. Simulate the behavior of the group.

Exercises

Exercise 8.– *Flocking*

See the correction video at https://youtu.be/g4X24h9yZAI

We consider $m = 20$ robots described by the following state equations:

$$\begin{cases} \dot{x}_i &= \cos\theta_i \\ \dot{y}_i &= \sin\theta_i \\ \dot{\theta}_i &= u_i \end{cases}$$

The state vector is $\mathbf{x}(i) = (x_i, y_i, \theta_i)$. These robots can see all other robots, but are not able to communicate with them. We want that there robots behave as a flock as illustrated by Figure 5.1. Basic models of flocking behavior are controlled by three rules of Reynolds: the *separation* (short range repulsion), the *alignment* and the *cohesion* (long range attraction). Using a potential based method, find a controller for each robot to obtain a flock.

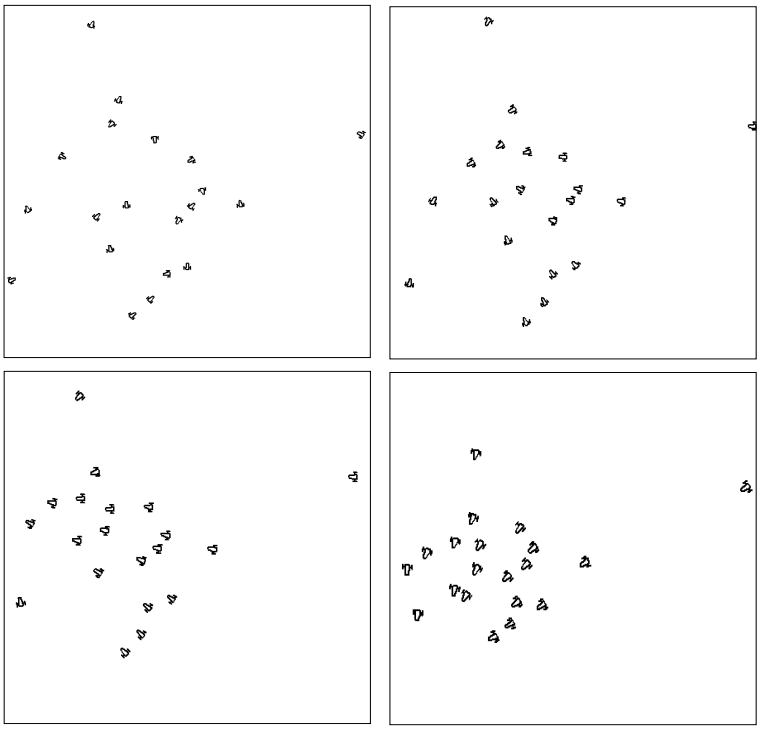


Figure 5.1: Illustration of the flocking behavior from a random initialization

---

EXERCISE 9.– *Consensus*

See the correction video at https://youtu.be/m5WZKFrFmeM

Let us consider m robots $R_1, \ldots, R_m$, as represented by Figure 5.2. These robots are all described by the following state equations:

$$\begin{cases} \dot{x}_1 &= x_4 \cos x_3 \\ \dot{x}_2 &= x_4 \sin x_3 \\ \dot{x}_3 &= u_1 \\ \dot{x}_4 &= u_2 \end{cases}$$

where $\mathbf{p} = (x_1, x_2)$ is the position of the robot, $\theta = x_3$ is the heading, and $v = x_4$ is the speed. The input vector is $\mathbf{u} = (u_1, u_2)$. The state for R_i will thus be denoted by $\mathbf{x}_i = (\mathbf{p}_i, \theta_i, v_i)$.

We assume that

- Each robot is able to see all other robots, *i.e.*, the robot R_i knows the bearing vector $\tilde{\mathbf{p}}_{ij}$ (in the R_i frame), and the heading difference $\tilde{\theta}_{ij} = \theta_j - \theta_i$ with respect to any other R_j, $j \neq i$.
- Each robot R_i is able to measure its own speed v_i
- The robots do not measure neither their position nor their heading, *i.e.*, they have no GPS and no compass

- The robots are identical and should thus enclose the same controller
- The robots are distinguishable, *i.e.*, they know what is the number of the robots they see.

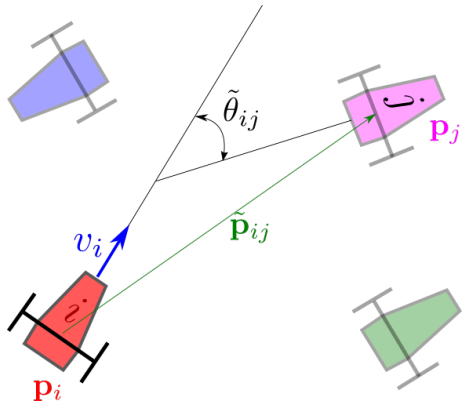


Figure 5.2: The four robots have to find a consensus in order to rotate around the same point. The robot R_i measures the bearing vector $\tilde{\mathbf{p}}_{ij}$ (in its own frame), and the angle $\tilde{\theta}_{ij}$

We want to find a controller to be implemented in each robot so that the swarm rotates forming a perfect circle. To reach our goal, we propose to a controller based on the two following rules :

- *Long range attraction.* Each robot R_i is attracted by an artificial hook at the back of R_{i+1} (where R_{m+1} corresponds to R_1), as illustrated by Figure 5.3. The corresponding artificial force is proportional to r, the distance between the two robots.
- *Close-range repulsion.* Each robot is repulsed by all other robots with an artificial force in $\frac{1}{r^2}$.

After a transition period, a consensus can be reached. Show on a simulation with 6 robots reaching a consensus where the robots form a perfect circle.

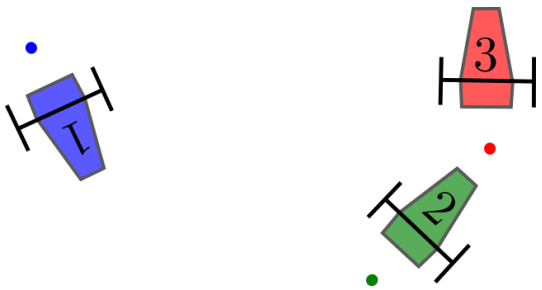


Figure 5.3: R_1 follows R_2, R_2 follows R_3,R_3 follows R_1. The artificial hooks are represented by the points at the back of each vehicle

---

EXERCISE 10.– *Platooning*

See the correction video at https://youtu.be/nv2utdAzesk

We consider $m = 10$ robots turning on a circular road of circumference $L = 100$m and with radius $r = \frac{L}{2\pi}$. Each robot $\mathcal{R}_i$ satisfies the following state equations

$$\begin{cases} \dot{a}_i &= v_i \\ \dot{v}_i &= u_i \end{cases}$$

The state vector is $\mathbf{x}(i) = (a_i, v_i)$ where a_i corresponds to the position of the robot and v_i to its speed. Each robot $\mathcal{R}_i$ is equipped with a radar which returns the distance d_i to the previous robot $\mathcal{R}_{i-1}$ and its derivative $\dot{d}_i$, as illustrated by Figure 5.4.

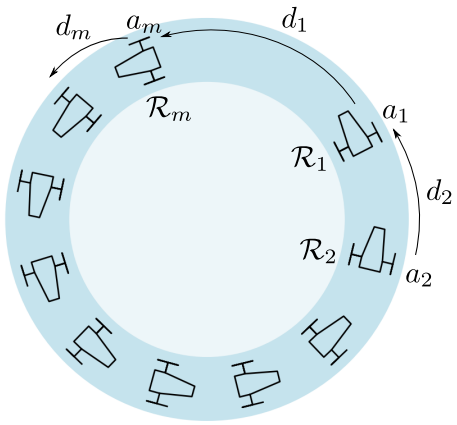


Figure 5.4: Platooning on the circle

1) Write the expression of the observation function $\mathbf{g}(a_{i^-}, a_i, v_{i^-}, v_i)$ which returns the vector $\mathbf{y}(i) = \left(d_i, \dot{d}_i\right)$. In this formula, $i^- = i - 1$ if $i > 0$ and $i^- = m$ if $i = 0$. Indeed, since the road is circular, the robot $\mathcal{R}_1$ follows the robot $\mathcal{R}_m$.

2) Propose a proportional and derivative control so that the robots will get a uniform distribution and go at a speed equal to $v_0 = 10ms^{-1}$. Check with a simulation.

3) In case of stability, prove theoretically that when the steady behavior is reached, all robots have a speed equal to v_0 and they are uniformly distributed.

4) Prove the stability of the system for 4 robots.

5) Provide a simulation with 10 robots.

Bibliography

[1] John W. Lloyd. *Foundations of Logic Programming.* Symbolic Computation. Springer, 2nd edition, 1987.

[2] A. Colmerauer. An introduction to prolog III. *Commun. ACM*, 33(7):69–90, 1990.

[3] Frédéric Benhamou and Touraïvane. Prolog IV : langage et algorithmes. In Jean-Jacques Chabrier, editor, *JFPLC'95, IVèmes Journées Francophones de Programmation en Logique & Journée d'étude Programmation par Contraintes et applications industrielles, 17-19 mai 1995, Dijon, France*, pages 51–64, 1995.

[4] Hans van Ditmarsch, Wiebe van der Hoek, and Barteld Kooi. *Dynamic Epistemic Logic*, volume 337 of *Synthese Library.* Springer, 2007.

[5] Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography.* CRC Press, Boca Raton, FL, USA, 1996.

[6] Nancy A. Lynch. *Distributed Algorithms.* Morgan Kaufmann, 1996.