UNIVERSITÀ DI BOLOGNA

School of Engineering

Master Degree in Automation Engineering

Distributed Control Systems

COVERAGE CONTROL FOR COOPERATIVE MULTI-ROBOT NETWORKS

Professor: **Giuseppe Notarstefano**

Students:
Donato Brusamento
Mattia Micozzi
Guido Carnevale
Lorenzo Draghetti

Academic year 2018/2019

Abstract

The aim of the project hereby presented was to transpose the algorithms described in the research paper [1] into a proper Matlab environment suitable for simulation and results analysis.

The aforementioned algorithms has been simplified in order to adapt to both an offline, sequential approach (from now on called **offline**,or **centralized**) and to a parallel (single machine), asynchronous one (from now on called **online**,or **distributed**).
In both approaches, the simulation implements a simple proportional closed loop control, but while in the former it is all done inside a centralized *for loop*, in which informations about each agent is readily available to all others, in the latter we tried to be more faithful to the original formulation, in which every agents has to exchange with other agents informations about each other's positions in a timely manner, while being "deaf" (to a certain extent) in-between successive communications.

To achieve all of this we've made use of Matlab's *Parallel Computing Toolbox* to simulate the independent agents as communicating threads.
In the distributed version of the simulation we also implemented a visual comparison between the two approaches so to show how they bring slightly different results.

Contents

List of Figures

Introduction

A mobile wireless sensor network (MWSN) can be defined as a wireless sensor network (WSN) in which the sensor nodes are mobile. For example the nodes can be wheeled robots scattered in a given area.

MWSNs are an emerging field of research in contrast to their well-established predecessor. They are way more flexible than static sensor networks as they can be deployed in any scenario and deal with rapid topology changes due to node failures or new added sensors. In general each node consists of a radio transceiver, a microcontroller powered by a battery and one or more sensors to detect certain properties of the environment. [2]
Typical applications of this kind of network are environment monitoring, surveillance, search and recovery operation, exploration.
A problem that can arise in this context is the optimization of coverage of a given area knowing the density distribution function, defined on that area, of a given property we would like to measure. The objective of this project was to implement a distributed and asynchronous algorithm to solve this problem, following the method proposed in the research paper [1].

The considered framework, as suggested by the aforementioned paper, is the following: n mobile sensor-robots modelled as simple integrators strewn on an area given by a convex polytope defined in $\mathbb{R}^2$; on this area a 2-dimensional distribution $\phi : Q \to \mathbb{R}_+$ is defined to represent the density of the feature that nodes have to measure.
Each node has sensing (can locate the other nodes positions) and/or communication capabilities.

Conceptually this algorithm runs iteratively 2 subroutines: one, called "Adjust-sensing radius algorithm", to compute the Voronoi cell of each sensor, and another one, called "Monitoring algorithm" to check if the computation of the Voronoi cell has to be updated because of significant changes in the network (variation of nodes positions, node failures,...).

We implemented this procedure in Matlab in 2 ways:

1. in a centralized sequential way
2. in a distributed parallel fashion, reading and writing files (stored in a master server) to realize communication among nodes.

These two implementations and the differences between them are well-explained and analysed in this report
In particular, the first chapter regards the problem set-up and description of proposed solutions; the second chapter contains the analysis of obtained results.

Motivations

The algorithms and simulations have been implemented in Matlab because of its (relatively speaking) simplicity and useful add-ons.
In particular, thanks to the graphical tools readily made available we could also create a real time results visualization to accompany the simulations, which is well suited to analyse the behaviour of both algorithm, compare them, and easily fix or add missing features.

Contributions

We believe that, thanks to this project, we created a starting point for building and implementing on real MWSN the proposed control algorithms; while further steps have to be taken in the direction of optimizing the code for embedded platforms (and simulations as well), the main idea of the paper used as a guideline, and its efficiency, have been demonstrated through our framework.

Chapter 1

Problem set-up and solutions

First-chapter for problem set-up and description of the implemented solution.

1.1 Setup

1.1.1 Theoretical notions

The algorithm proposed by the main paper is **Lloyd's gradient descent**, which guarantees the convergence to an optimal solution intuitively given by the trade-off between each node's coverage (position) and relative degradation of sensing capabilities.
A main point to be considered is that this solution makes use of the concept of Voronoi partitions.
Given a subspace S in $\mathbb{R}^N$ (typically a plane) and a set of n points $P = \{p_1, ..., p_n\}$ belonging to this subspace we call Voronoi partition a partition such that there is an associated region for each point and this region consists of all points closer to that point than any other point. We call these regions Voronoi Cells.
Let's see an example of Voronoi Partition of a plane:

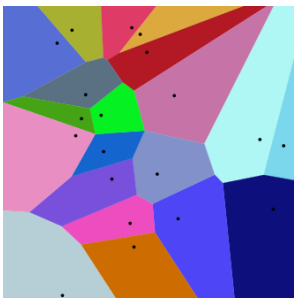

Figure 1.1: Voronoi Cells of 20 points on a plane

Considering a convex polytope Q in $\mathbb{R}^N$ and n sensors, for the sensor positions $P = \{p_1, ..., p_n\}$ the optimal partition of Q is the Voronoi partition $V(P) = \{V_1, ..., V_n\}$, where V_i is the Voronoi cell of the i-th sensor:

$$V_i = \{q \in Q | \, \|q - p_i\| \leq \|q - p_j\|, \forall j \neq i\}$$

Intuitively speaking, each V_i represents the region of space in which the respective sensor i is performing its task.

Citation [1]

Because of noise and loss of resolution we assume also that the sensing quality of a sensor placed at p_i wrt a point q decreases as much the distance between p_1 and q is large. Then once we have computed the Voronoi Partition we have also to find for each Voronoi Cell the point in which the sensing degradation is minimized in order to move the sensor in this point. Obviously this motion modifies the situation and may change the Voronoi Partition. As in the paper we have modeled the sensing degradation as the squared distance between the sensor location and the considered point. Under this assumption we have that for each Voronoi Cell the point in which the sensing degradation is minimized is the so called generalized centroid C_v of the Voronoi Cell namely the center of mass (generalized mass M_v) that we find considering the distribution density function of the variable that we want to measure as a mass density function p.

$M_v = \int_V p(q)\, dq$

$C_v = \frac{1}{M_v} \int_V qp(q)\, dq$

The adopted alogrtihm **Lloyd's gradient descent** guarantees to reach a global convergence under the given assumptions.

1.1.2 Adopted assumptions

Although the code was implemented with the objective in mind of staying as much faithful to the paper as possible, some assumptions had to be made in order to work with the simulations running on a single machine, especially considering that some pieces of code (e.g. the computation of Voronoi cells) tend to slow them down quite a bit, and should then be optimized during further developments.

In particular, with respect to the "ideal" version of the algorithm proposed in the paper [1], the assumptions made are

1. The *sensing* of other agents is not implemented by a specific communication protocol; in the distribution version case, text files were written and read in a similar fashion to *POSIX pipes*, and the only reason preventing an immediate sensing is the lock put to files during mutual r/w.
   This, as a side effects, also "forces" the lack of synchronism between threads.

2. The *timing*, as in the scheduling of different subroutines like monitoring and sensing, is not implemented, as it would require real platforms with only few unique threads running so to not introduce large discrepancies in convergence speeds.
   This is not only due to (rather severe) constraints of horsepower available on the used machines, but also to the lack of ready to use, POSIX-like scheduling primitives in Matlab's *Parallel Computing Toolbox*.
   On top of that, while this would definitely be an issue in the real world implementation of the algorithm, for the purpose of this project (i.e. demonstration and simulation of the algorithm and creation of essential tools for its correct implementation), the proposed version suffices in its goals, and the following point explains the trade-off that was made in accordance with the professor's directions.

3. For the same reason, while the correct calculation of neighbours weights is implemented and also displayed in the Offline algorithm, in the Distributed version agents do not wait for *events* in order to change control, but rather they first reach the last computed region's centroid, and only then they sense others' position to update their own cell and reference centroid.
   However, should the need arise, the primitives and functions aiding in the event generation are already available in Ver. A of the code, as mentioned.

1.2 Inner Workings

1.2.1 Offline algorithm (A)

As in fig. 1.2...

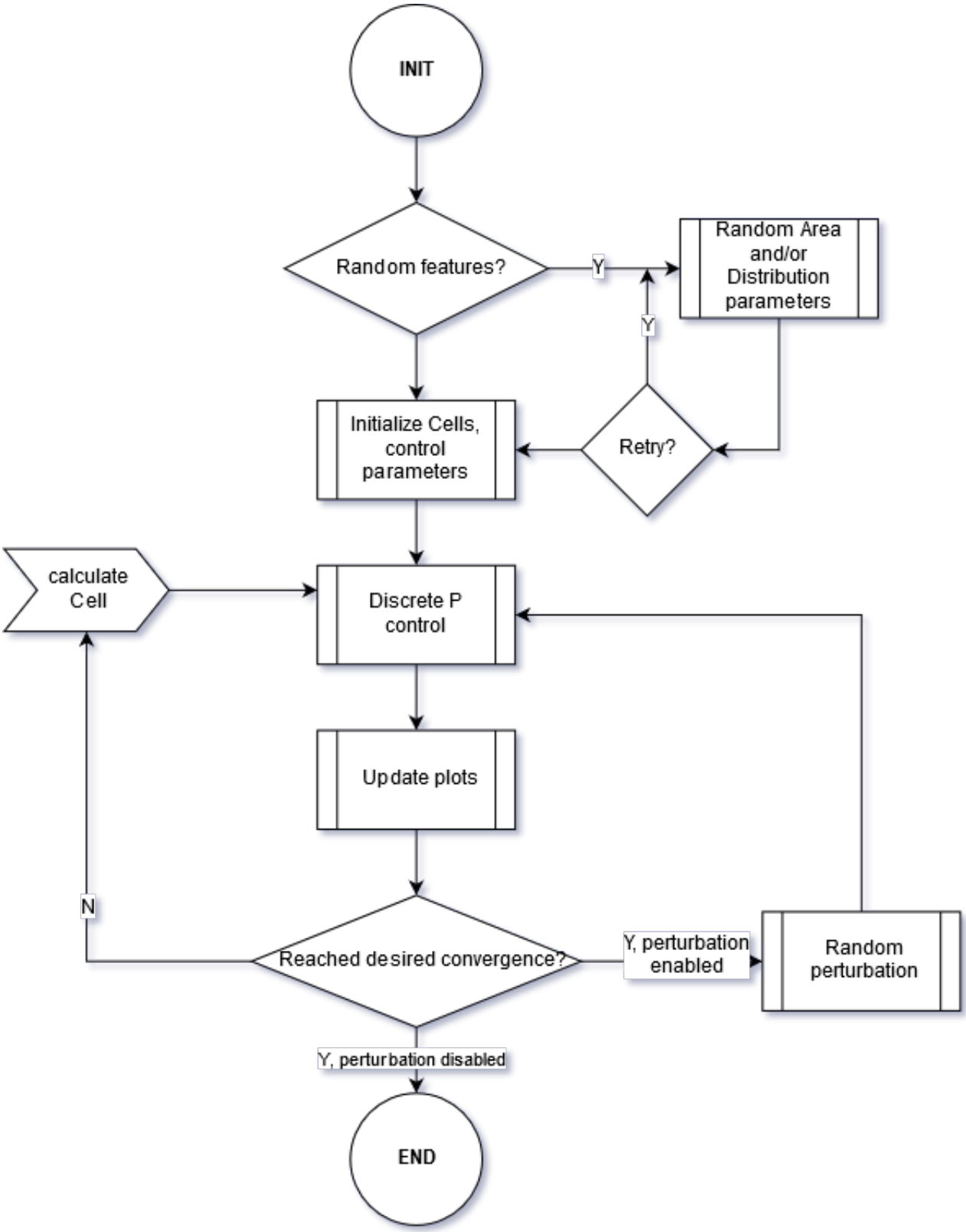

Figure 1.2: Algorithm A flow

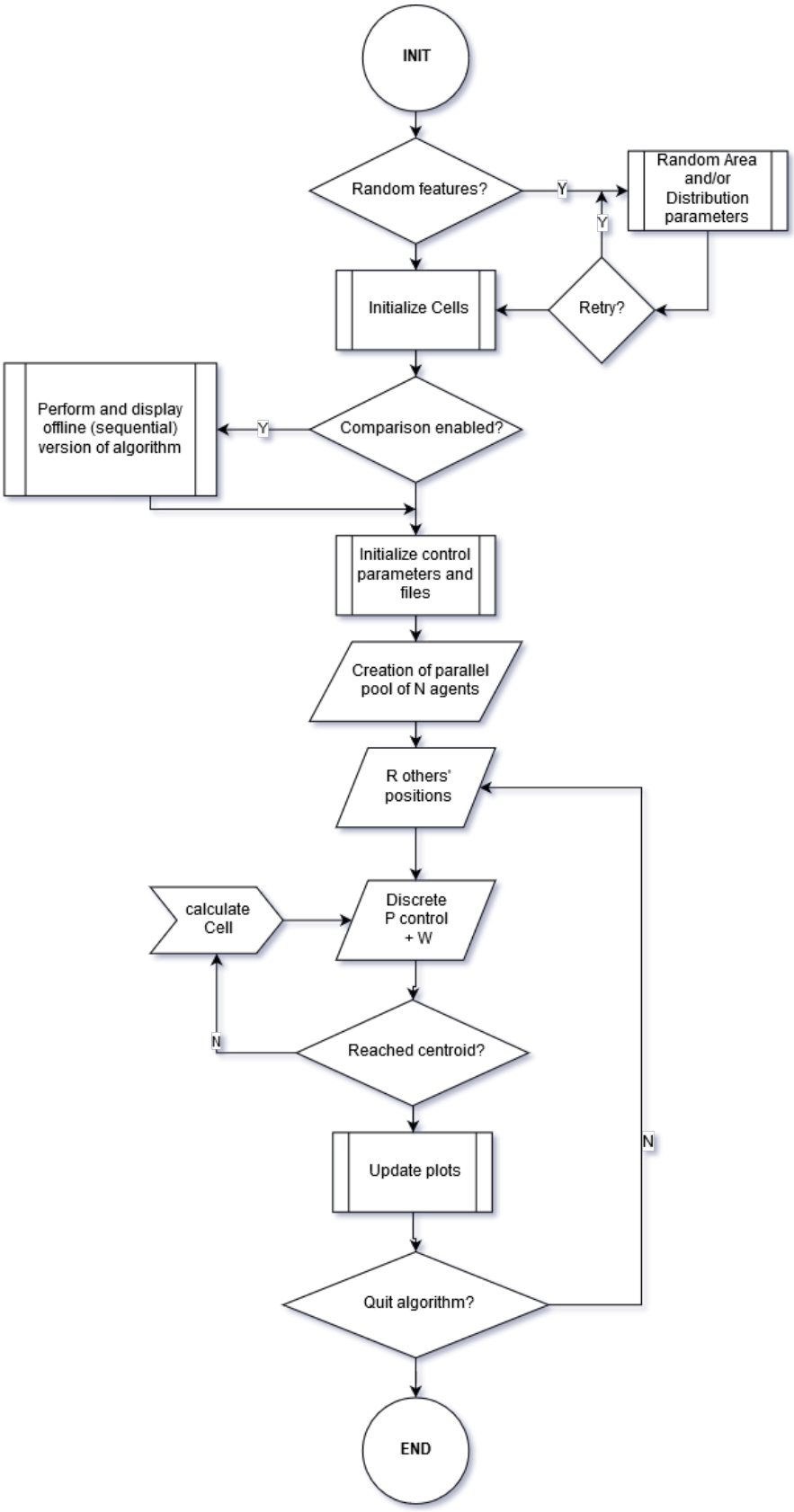

Figure 1.3: Algorithm B flow

1.2.2 Online algorithm (B)

1.2.3 Utils

calculateCell

sliceCell

genMCGauss

Chapter 2

Results analysis

2.1 Prerequisites

For the correct functioning of the simulations, the following Matlab Toolboxes and FEX files should be installed:

- Parallel Computing Toolbox
- Mapping Toolbox
- labelpoints, by Adam Danz
- Fast and Robust Curve Intersections, by Douglas Schwarz

2.2 Simulation benchmarks

In both approaches, several parameters can be adjusted that affects precision (and conversely, speed of simulations), namely:

- number of agents;
- query points density for distribution computation;
- proportional gain of the control;
- convergence threshold;
- etc.

In addition to that, the generation of the area to cover, of the agents and of the distribution can be set to randomly generate at each start of simulation (see fig. 2.1, fig. 2.2).

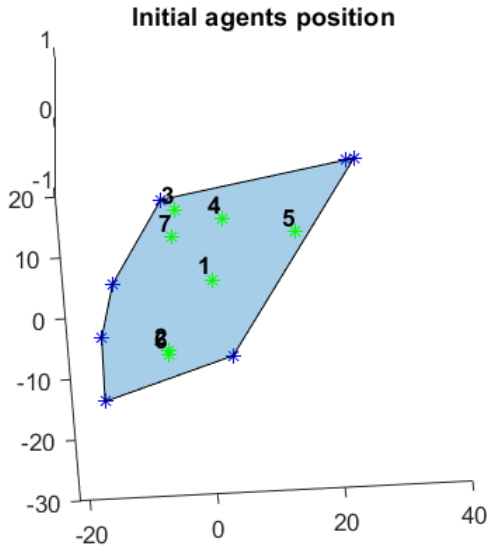

Figure 2.1: Random agents spawning

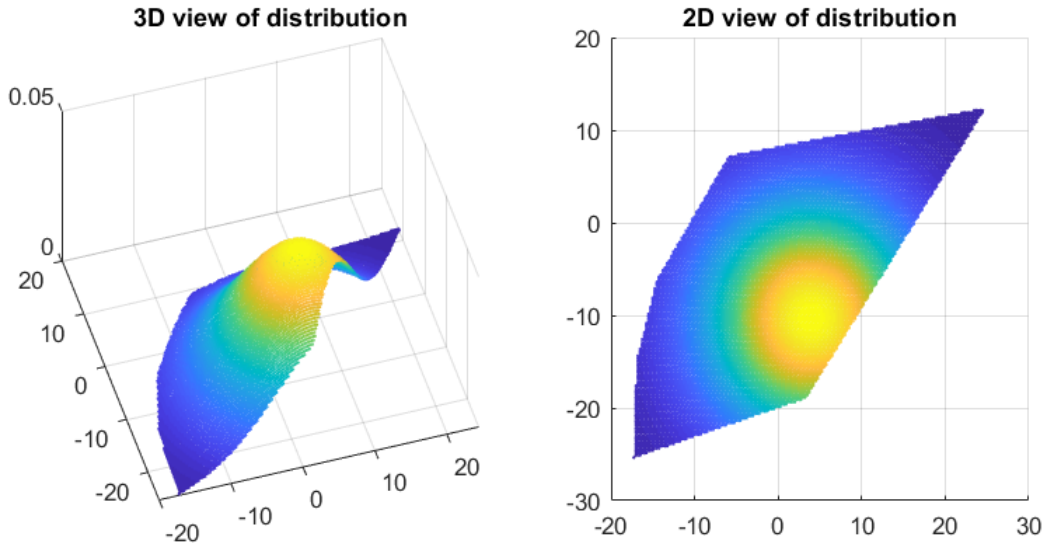

Figure 2.2: Random 2D normal distribution

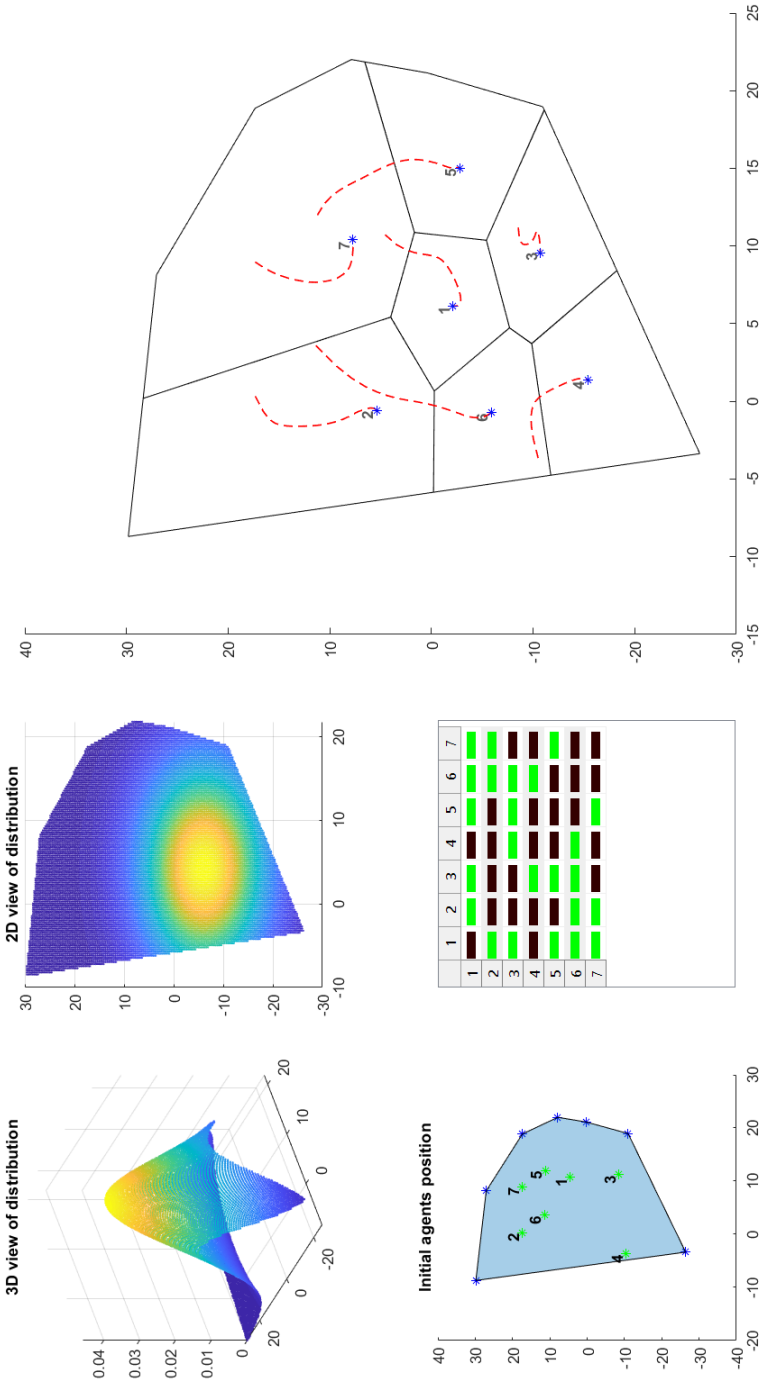

Figure 2.3: Results with 7 agents

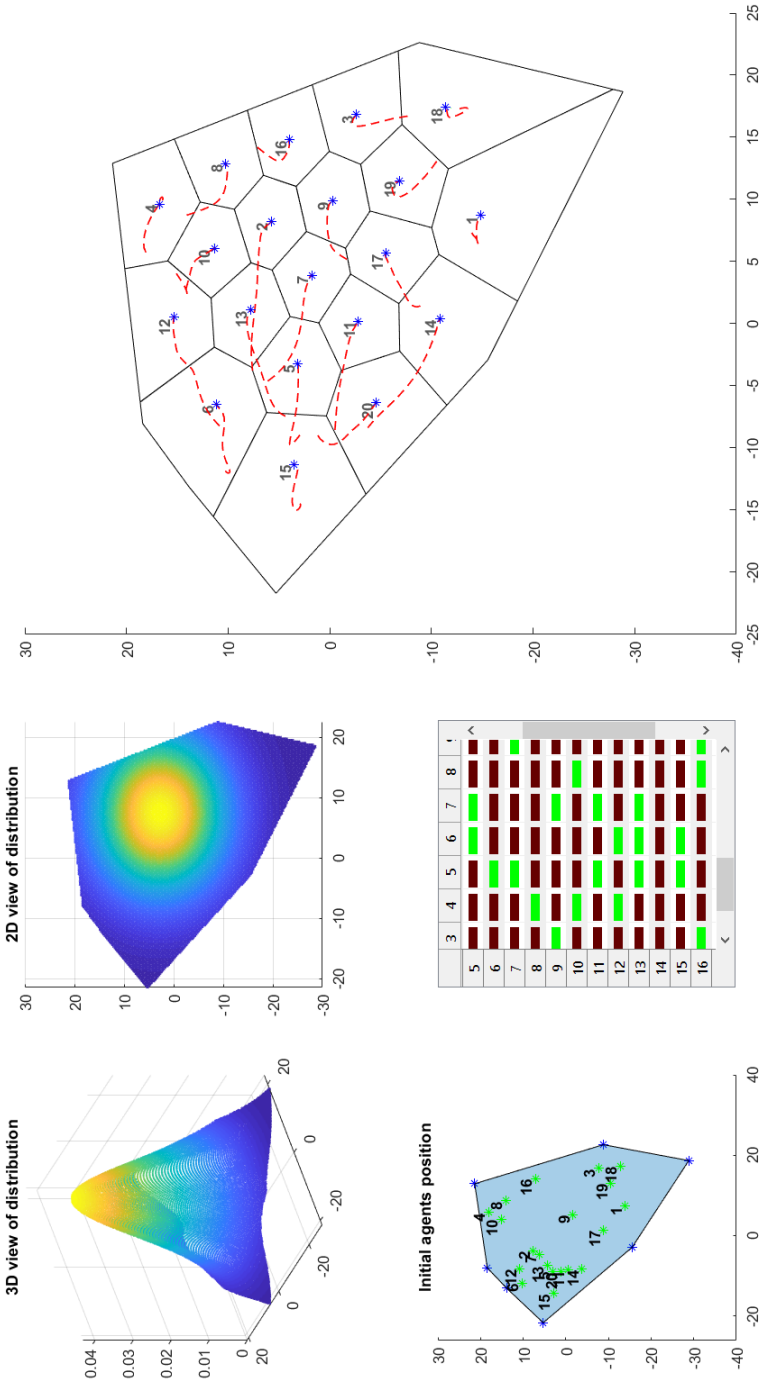

Figure 2.4: Results with 20 agents

2.3 Offline algorithm (A)

2.4 Online algorithm (B)

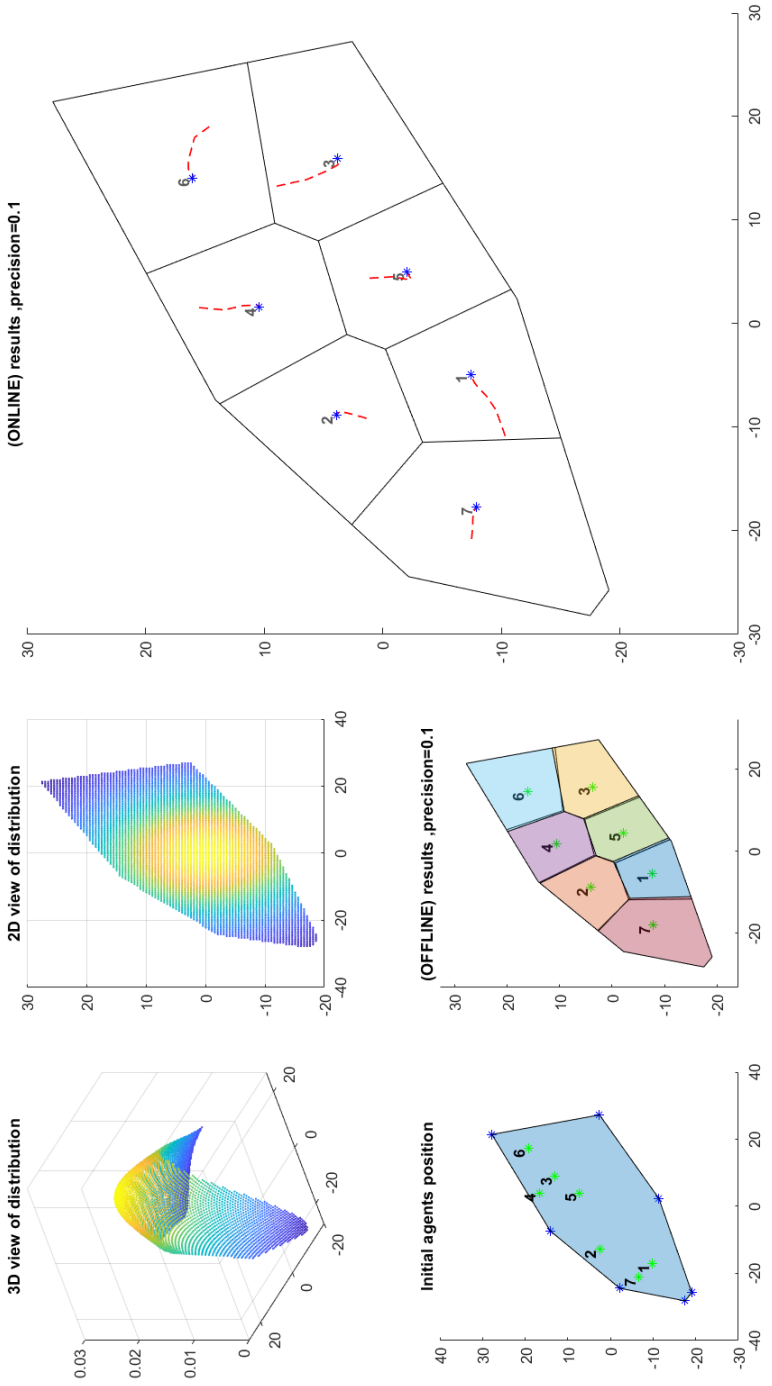

Figure 2.5: Results with 7 agents

Conclusions

The obtained results seem to be very good and then we think that this work may be a good starting point to realize a real Coverage Control for Cooperative Multi-Robot Networks.

We have negletted a little bit optimization problems from the code efficiency point of view, nonetheless the execution times seem to be reasonably short.
So we think that the algortihm can be improved in terms of code optimization and task communication efficiency.

Bibliography

[1] J. Cortés, S. Martìnez, T. Karatas, F. Bullo. "Coverage Control for Mobile Sensing Networks" IEEE Transactions on Robotics and Automation, Vol. 20, No. 2, April 2004

[2] T. Hayes and F.H. Ali. "Mobile Wireless Sensor Networks: Applications and Routing Protocols" Handbook of Research on Next Generation Mobile Communications Systems. 2016.