Distributed Partitioning Algorithm with application to video-surveillance

Authors:
Inés Paloma
Carlos Saiz

Supervisor:
Ruggero Carli

June 30, 2015
Contents

1 Introduction 1

2 Problem formulation 2

3 Line partitioning 3
  3.1 Synchronous case 3
  3.2 Asynchronous 4
    3.2.1 Symmetric 4
    3.2.2 Asymmetric 6

4 Monte Carlo Simulations 9

5 Line partitioning: Proof of convergence 10
  5.1 Stability of matrix A 13

6 Alternative way to proof convergence 17
  6.1 Proof of convergence 17
  6.2 Convergence rate 19

7 Voronoi partitioning of an area 21
  7.1 Synchronous case 21
    7.1.1 Algorithm 1 22
    7.1.2 Algorithm 2 23
  7.2 Asynchronous case 24
  7.3 Density distribution 25

8 Simulation results 27
  8.1 1-Dimensional Case 27
    8.1.1 Convergence rate 30
  8.2 2-Dimensional Case 30
    8.2.1 Different density functions 33
    8.2.2 Convergence rate 33

9 Conclusions 34
1 Introduction

How many times have we wanted to control an area and the video-surveillance system does not have the appropriate properties? Nowadays, the video-surveillance has become a responsibility by the necessity to patrol a specific area. Even if there are available HD cameras that can detect movements and heat fluxes, the real problem is to have a way of controlling the behaviour of the cameras in order to provide a good and autonomous video-surveillance system. Only exploiting communication among cameras, the system should cover all the area, not allowing to an object to cross the area without being detected. Also, a very important aspect to consider is the speed of the algorithm to coordinate the cameras in order to converge to an optimal covering, where each camera has to take care of a specific area.

The coverage and partitioning of an area is one fundamental task that needs to be accomplished in order to execute more complex tasks, for example the application in video-surveillance. The appealing idea is that a group of robots have the ability to communicate among them in order to cover and control the area. Although we will be focusing on camera-surveillance the algorithm may be used in other fields, such as vacuum and security robots.

The aim of this project is to develop an algorithm that allows us to partition and cover an area of interest. This algorithm has to be robust and adaptive to possible changes in the controlled area. Another important aspect to take into account is the possibility to add or remove new cameras, or considering the variation of the dimensions of the area.

The challenge that we want to address is to provide an algorithm close to reality. In particular, we will focus on the communication between cameras explaining different protocols and their behaviours. In order to proceed step by step, we will firstly create a synchronous algorithm where the robots interact simultaneously and then we will add constraints in order to get to an asynchronous and asymmetric algorithm, which is the most realistic. In the synchronous system, the robots are all together communicating at every time instant so they are updating data. Conversely, in the asynchronous protocol only a pair of robots interact at a specific time, so maybe they have the necessity to communicate and they are not able to do it.

There has been considerable effort in the analysis of minimizing the time to converge to an optimal solution and also we want to exploit the concept of centroidal Voronoi partitions to optimally divide the monitored area.

Finally, it will be studied the 2-Dimensional case where it will be possible to simulate a real case. We will provide a video where we show how the algorithm works and how the agents are communicating to calculate the area covered by each one of them.
2 Problem formulation

One-dimensional case

We want to distribute a specific number of agents, $N$ in a line with a determined length $L$. We denote the agents with the letter $i \in (1,...,N)$ and every one has associated a middle point, which is the position of the agent $m_i$ and two edges, right and left side, that define the area covered by the agent. They are called $r_i$ and $l_i$. Each agent, except from the first and last agents, will have two neighbours, the agents that along the line, proceed and follow him. The algorithm will perform a fixed number of iterations, $k \in (1,...,K)$ and will return the final position of each agent $m_i(k)$. Using the theoretical position $tp_i$, which is the position that each agent would have if there were infinite iterations, we can obtain the error that will have to be smaller than a fixed value called $\epsilon$.

Two-Dimensional case

We want to distribute a specific number of agents, $N$ in a space with a determined area $A_T$, defined by the width, $w$, and the height, $h$. We denote the agents with the letter $i \in (1,...,N)$ and every one has associated a position which will be the centroid of the area that it has to cover, called $c_i$ and it has the components $c_x$ and $c_y$. The area covered by each agent is called $a_i$. The number of iterations is undefined and it depends on the number of agents and the initial position of each of them.
3 Line partitioning

In this section we review the problem of monitoring a one-dimensional environment of finite length with \( N \) agents, where \( N \) is a finite number. Each agent has associated an finite interval and according to this, it is possible to have three different situations. One happens when there is some space which is not covered by any agent, so blank spaces are created, the opposite is the overlapping that happens when an area is covered by more than one agent at the same time. Otherwise, if there is not overlapping neither blank spaces and each agent is covering a different interval, the sum of the intervals of each agent is \( L \).

The agents are labelled 1 through \( N \), and for the sake of simplicity, we assume the following characteristics:

i. Each agent has the same memory,

ii. Each agent will control a part of the environment of the same length.

3.1 Synchronous case

In this case, as it is a synchronous case, all the agents are going to move and update their position at the same rate. Therefore, in every step of the algorithm every agent receives and send information to its neighbours and updates its position considering them.

We assume the following hypothesis:

i. All the agents are synchronized, each one sends and receive information at the same time,

ii. All the agents are communicating among them.

In our analysis, we assume that there is no overlapping either blank space between the areas controlled by the agents and the boundaries of the areas satisfy the following constraints:

\[
\begin{align*}
    r_i &= l_{i+1} & \forall & i \ (1, \ldots, N - 1) \\
    l_1 &= 0 \\
    r_N &= L
\end{align*}
\]  

(3.1.1)

Algorithm 1 Synchronous Symmetric Case

<table>
<thead>
<tr>
<th>Require: ( N, K, L, m_i(0) \ \forall i \in (1, \ldots, N) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1: for ( k = 1, \ldots, K ) do</td>
</tr>
<tr>
<td>2: for ( i = 1, \ldots, N ) do</td>
</tr>
<tr>
<td>3: Perform the equations (3.1.1) and (3.1.2) to calculate the middle point between each pair of agents which is going to be the new edge.</td>
</tr>
<tr>
<td>4: Perform the equation (3.1.3) to update the position of each agent with new edges</td>
</tr>
<tr>
<td>5: end for</td>
</tr>
<tr>
<td>6: end for</td>
</tr>
<tr>
<td>7: return ( m(K) )</td>
</tr>
</tbody>
</table>

Firstly, in every step, for each pair of agents it is required to calculate the middle point between them, which is going to be the new edge for the next step. After that, the position of each
agent is updated with the new edges.

Standard algebraic manipulations show that \( m_i(k), i \in \{1, ..., N\}, \) satisfy the following recursive equations

\[
\begin{align*}
r_i(k + 1) &= \frac{m_i(k) + m_{i+1}(k)}{2} \\
l_i(k + 1) &= \frac{m_i(k) + m_{i-1}(k)}{2}
\end{align*}
\] (3.1.2)

Recall the equation (3.1.1) to obtain the edges for the first and last agents and update the position of the each agent using the following equation.

\[ m_i(k + 1) = \frac{r_i(k) + l_i(k)}{2} \] (3.1.3)

Finally, it is possible to calculate the error between \( tp_i \) and \( m_i(k) \) which are explained before, it depends on the number of iterations. The higher is the number of iterations, the fewer will be the error.

\[ e(k) = \|tp - m(k)\|_\infty, \forall i(1, ..., N) \]

Where with \( \| \cdot \|_\infty \) we mean the infinite number of iterations.

### 3.2 Asynchronous

The asynchronous case, in contrast to the synchronous, is that the communication in every step is only between two agents who are selected randomly. Inside this case, we can distinguish between two different update systems, one symmetric and the other asymmetric.

#### 3.2.1 Symmetric

We would like to have only two agents interacting between them because this situation is more likely to the reality. It is an update system where the two agents who are communicating in the same step are both updating their position. This algorithm is going to be explained in the Algorithm 2.

The following hypothesis are considered:

i. Each agent can receive and send information at the same time,

ii. In every step both agents will interact to modify their positions.

Firstly, an agent is selected randomly as well as one of his neighbours. In each step, this process will be repeated and depending on the agent selected, we distinguish between three different alternatives:

1. Considering that the agent selected is the 1st one, it can only interact with the 2nd one,

\[
\begin{align*}
c(k + 1) &= \frac{m_1(k) + m_2(k)}{2} \\
l_2(k + 1) &= c(k + 1) \\
r_1(k + 1) &= c(k + 1)
\end{align*}
\] (3.2.1)
Algorithm 2 Asynchronous Symmetric Case

Require: $N, K, L, m_i(0)$ \forall i (1,...,N)

1: for $k = 1,...,K$ do
2:     $i = \text{random}(N)$
3:     if $i = 1$ or $i = N$ then
4:         Perform the equations (3.2.1) and (3.2.2) when $i = 1$ or the equations (3.2.3) and (3.2.4) when $i = N$, so they interact with their only neighbours calculating the $m_i(k+1)$ and $c(k+1)$
5:     else
6:         Choose randomly one of their neighbours and perform the equations (3.2.5) and (3.2.6) to calculate with him their $m_i(k+1)$ and $c(k+1)$
7:     end if
8: end for
9: return $m_K$

The $c$ is a temporal variable used to store the new edge of each agent.

After that, it is calculated the new middle points of agents one and two.

\[
\begin{align*}
    m_1(k+1) &= \frac{r_1(k+1)}{2} \\
    m_2(k+1) &= \frac{l_2(k+1) + r_1(k+1)}{2}
\end{align*}
\]

(3.2.2)

2. Considering the agent selected is the last one:

\[
\begin{align*}
    c(k+1) &= \frac{m_{N-1}(k) + m_N(k)}{2} \\
    l_N(k+1) &= c(k+1) \\
    r_{N-1}(k+1) &= c(k+1)
\end{align*}
\]

(3.2.3)

To continue, it is necessary the computation of the new middle points of the last two agents

\[
\begin{align*}
    m_{N-1}(k+1) &= \frac{l_{N-1}(k+1) + r_{N-1}(k+1)}{2} \\
    m_N(k+1) &= \frac{l_N(k+1) + r_N(k+1)}{2}
\end{align*}
\]

(3.2.4)

3. Otherwise, before interacting it is randomly chosen one of its neighbours. In the following lines there is only explained when the agent selected interacts with his right side.
\[ c(k + 1) = \frac{m_{i+1}(k) + m_i(k)}{2} \]  
\[ l_{i+1}(k + 1) = c(k + 1) \]
\[ r_i(k + 1) = c(k + 1) \]  
\[ m_{i+1}(k + 1) = \frac{l_{i+1}(k + 1) + r_{i+1}(k + 1)}{2} \]  
\[ m_i(k + 1) = \frac{l_i(k + 1) + r_i(k + 1)}{2} \]

As in the other alternatives, compute of the new middle points,

Finally, these lines of code will be repeated until the number of steps previously fixed is reached.

3.2.2 Asymmetric

In this algorithm, we would like to get closer to reality and therefore considers that, just one of the two agents that communicate sends the information and the other receives it. Consequently, only the one who receives the information can move its position.

As in the symmetric sample, there are some initial hypotheses to consider:

i. Each agent can only receive or send at the same time,

ii. only the one who receives data can change its position.

Assuming that only one of the agents can move and modify its position, there are two possibilities; leaving blank spaces or having overlapping between the intervals that each agent has to cover.

![Figure 1: The interval of the agent who receives information is bigger than the one who sends it, so when he updates his position leaves a blank space between them.](image)

6
Algorithm 3 Asynchronous Asymmetric Case

Require: $N$, $K$, $L$, $x_i(0)$ $\forall i(1,...,N)$

1: for $k = 1,...,K$ do
2: $i = \text{random}(N)$
3: if $i = 1$ then
4: Perform equations (3.2.1) and (3.2.2) to calculate $m_i(k + 1)$ and $l_2(k + 1)$
5: if $l_2(k + 1) > r_1(k + 1)$ then
6: Perform the equation $l_2(k + 1) = r_1(k)$
7: else
8: Proceed with the equation (3.2.2) to update the new edge of the agent 2
9: end if
10: else if $i = N$ then
11: Perform the equations (3.2.3) and (3.2.4) to calculate $m_i(k + 1)$ and $r_{N-1}(k + 1)$
12: if $l_N(k + 1) > r_{N-1}(k + 1)$ then
13: Perform the equation $r_{N-1}(k + 1) = l_N(k)$
14: else
15: Proceed with the equation (3.2.4) to update the new edge of the agent N-1
16: end if
17: else if $i \neq 1$ and $i \neq N$ then
18: Choose randomly the left or the right agent and perform the equations (3.2.5) and (3.2.6) to calculate with him $m_i(k + 1)$ and the new edges
19: if $l_i(k + 1) > r_{i-1}(k + 1)$ (resp. $l_{i+1}(k + 1) > r_i(k + 1)$) then
20: Perform the equations (3.2.7) therefore the right (resp. left) edge of the receiver agent will be the same as the left (resp. right) edge of sending agent
21: else
22: Proceed with the equation (3.2.6) to update the position of the agent who receives information
23: end if
24: end if
25: Calculate $m_i(k + 1)$ and $m_{i+1}(k + 1)$ or $m_{i-1}(k + 1)$
26: end for
27: return $m_K$

Figure 2: The interval of the agent who receives information is smaller than the one who sends it, so when he update his position covers a perimeter of the other agent.
One possibility can occur when the interval of the agent who receives information \((i + 1)\) is bigger than the one who sends it \((i)\) due to the middle point between them will be located in the interval of the second one. Consequently, there will appear a blank space, not covered by any agent. In the Figure 1 it is shown an example:

The other possibility, as it is seen in the Figure 2, is the opposite, happens when the interval of the agent who sends information \((i)\) is bigger than the one who receives it \((i + 1)\). So in this situation there will be an interval covered by both agents.

Considering these possibilities, as it is interesting to have the perimeter covered in every moment, blank spaces should not appear. In order to avoid them, the algorithm will contain a detection command to pass to the next step without causing changes.

The process to start the algorithm is the same as the synchronous but including an “if” condition to avoid blank spaces.

Therefore, if a blank space is not going to be produced, the method is the same as the synchronous sample. Otherwise, the new “if” condition will proceed in the following mode:

1. If agent who sends information is agent 1
   \[ l_2(k + 1) = r_1(k) \]

2. If agent who sends the information is the last agent
   \[ r_{N-1}(k + 1) = l_N(k) \]

3. Otherwise. Depending on the neighbour chosen, there are two ways to compute the new edges.
   \[
   \begin{align*}
   r_{i-1}(k + 1) &= l_i(k) \\
   l_{i+1}(k + 1) &= r_i(k)
   \end{align*}
   \]

To conclude, the algorithm of the one-dimensional environment that is closer to reality is the last one because it contemplates all possible choices and acts accordingly.
4 Monte Carlo Simulations

Monte Carlo simulations are a broad class of computational algorithms that rely on repeated random sampling to obtain numerical results. In our algorithm we want to compute the evolution of the relative error between the final position of each agent after a predefined number of iterations and the theoretical position of each agent.

We are doing the Monte Carlo simulations for the synchronous and the asynchronous case, and we have computed the logarithm of the mean of the error. The number of simulations that we have done is 200, each one with 200000 iterations.

Algorithm 4 Monte Carlo Simulations

\begin{algorithm}
\begin{algorithmic}
\Require $N, NMC, L, m_i(K) \ \forall i (1, \ldots, N)$
\For{$j = 1, \ldots, NMC$}
\State Execute algorithm synchronous
\State Execute algorithm asynchronous symmetric
\State Execute algorithm asynchronous asymmetric
\EndFor
\For{$i = 1, \ldots, N$}
\State Perform the equations (4.0.8) in order to calculate the error between the local and the theoretical position of each agent
\EndFor
\State Perform the equation (4.0.9) to do the mean of the error of all the agents
\State \Return $\overline{err}(K)$
\end{algorithmic}
\end{algorithm}

The process to compute the mean error with Monte Carlo simulations is the following.

First, it is necessary to compute the error of each agent, where $p_1$ is the theoretical position of the first agent and $err_i$ is the error associated to each agent.

\begin{equation}
\begin{align*}
  p_1 &= L/2N \\
  tp_i &= i \times p_1 \\
  err_i(K) &= |tp_i - m_i(K)|
\end{align*}
\end{equation}

Then, in order to calculate the mean error, $\overline{err}(K)$ of all the agents it is necessary to compute the following:

\begin{equation}
\overline{err}(K) = \sqrt{\sum(err_i)^2}
\end{equation}

Finally, we obtain the plot of mean error and we can apply the same procedure to the different cases that we have done, Synchronous and Asynchronous algorithms. If we do the logarithm of the mean error in each case we can see that the slope of the line is directly related to the speed of convergence. So, if ordered them by speed of convergence we see that the synchronous is faster than the symmetric asynchronous, and these two outperform the asynchronous asymmetric. The reason because the synchronous is faster is because in every iteration all the agents are communicating but we have had to do strong assumptions, why in the asynchronous case the implementation is easier but the convergence is slower.
5 Line partitioning: Proof of convergence

With the assumptions made before, we wanted to verify that the hypotheses are correct and the theorem that we propose is converging to zero. Also, we wanted to express the new position of each agent as a stable matrix multiplied for the actual position plus a constant. The following equation shows what is has been explained.

In this proof the position of each agent will be called as \( x_i(k) \) instead of \( m_i(k) \)

\[
x(k + 1) = A \cdot x(k) + u, \quad x \in \mathbb{R}^N, \quad A \in \mathbb{R}^{N \times N}, \quad u \in \mathbb{R}^N
\]

As it is known, each agent has associated two edges, right and left, and a middle point which represents the location of the agent in every step of the algorithm.

Before continuing the demonstration, there are some concepts to clarify. Firstly, we would like to recall how to calculate the middle point of each agent and then we will try to express the new position of each agent as a function of its current position and the agents of its sides.

a. Calculate the current position of each agent

\[
x_i(k) = \frac{l_i(k) + r_i(k)}{2} \quad \forall \ i (1, \ldots, N)
\]

When \( i \) equals to 1 or \( N \), recall the equation (3.1.1).

b. Expressing the new position in the next step as a function of the current positions

\[
\begin{align*}
x_1(k + 1) &= f(x_1(k), x_2(k)) \\
x_i(k + 1) &= f(x_{i-1}(k), x_i(k), x_{i+1}(k)) \\
x_N(k + 1) &= f(x_{N-1}(k), x_N(k))
\end{align*}
\]

Now, we are prepared to proceed to the demonstration and to corroborate that the algorithm converges, we follow the next steps.

Firstly, compute the new left and right edges of each agent including the initial conditions

\[
\begin{align*}
r_i(k + 1) &= \frac{x_i(k) + x_{i+1}(k)}{2} \\
l_i(k + 1) &= \frac{x_i(k) + x_{i-1}(k)}{2}
\end{align*}
\]

If it is possible to express the left and right edges of the new step, the new middle points will be the half-sum of the current left and right edges

\[
x_i(k + 1) = \frac{l_i(k + 1) + r_i(k + 1)}{2}
\]

Then, we can include the equations of the 2nd step in the 3rd one.

\[
x_i(k + 1) = \left( \frac{x_i(k) + x_{i-1}(k)}{2} + \frac{x_i(k) + x_{i+1}(k)}{2} \right) \cdot \frac{1}{2} = \frac{x_i(k)}{2} + \frac{x_{i+1}(k)}{4} + \frac{x_{i-1}(k)}{4}
\]

Now, we will take into consideration the first and last agents because these steps differ from the general demonstration.
a. For the first agent

\[ r_1(k+1) = \frac{x_1(k) + x_2(k)}{2} \]
\[ x_1(k+1) = \frac{x_1(k) + x_2(k)}{4} \]

b. For the last agent

\[ l_N(k+1) = \frac{x_N(k) + x_{N-1}(k)}{2} \]
\[ x_N(k+1) = \frac{L + l_N(k+1)}{2} \]
\[ x_N(k+1) = \frac{L}{2} + \frac{x_N(k)}{4} + \frac{x_{N-1}(k)}{4} \]

Once all the alternatives have been studied we can write down the conditions of the system in a matrix and a vector of constants.

\[
A = \begin{bmatrix}
\frac{1}{4} & \frac{1}{4} & 0 & \ldots & \ldots & 0 \\
\frac{1}{4} & \frac{1}{2} & \frac{1}{4} & 0 & \ldots & \vdots \\
0 & \frac{1}{4} & \frac{1}{2} & \frac{1}{4} & \ldots & \vdots \\
\vdots & 0 & \frac{1}{4} & \frac{1}{2} & \ddots & 0 \\
\vdots & \vdots & \ddots & \ddots & \ddots & \frac{1}{4} \\
0 & \ldots & \ldots & 0 & \frac{1}{4} & \frac{1}{4} \\
\end{bmatrix}, \quad u = \begin{bmatrix}
0 \\
\vdots \\
\vdots \\
\vdots \\
0
\end{bmatrix}
\]

(5.0.10)

To demonstrate the effectiveness of the system obtained, it has to converge in an infinite time. In the following step we will calculate the final position of each agent in a generic way.

\[ x(k) \to \bar{x} \]
\[ \bar{x} = \begin{bmatrix}
\frac{L}{2N} \\
\frac{L}{2N} + \frac{L}{N} = \frac{3L}{2N} \\
\frac{3L}{2N} + \frac{L}{N} = \frac{5L}{2N}
\end{bmatrix} \rightarrow x_1 \]
\[ \rightarrow x_2 \]
\[ \rightarrow x_3 \]

Being L the length of the line, N the number of agents and i the desired agent; the position of each agent at the end will be

\[ x_i = \frac{(2i-1) \cdot L}{2N} \]

If the system converges, satisfy the conditions:

\[ \bar{x} : \quad \bar{x} = A \cdot \bar{x} + u \]
a. Corroboration of the convergence of the first agent:

\[ [A \cdot x + u]_1 = x_1 \implies \frac{1}{4} \cdot x_1 + \frac{1}{4} \cdot x_2 + 0 = x_1 \]

\[ \frac{1}{4} \cdot \frac{L}{2N} + \frac{1}{4} \cdot \frac{3L}{2N} = \frac{L}{2N} \]

\[ \frac{1}{8} \cdot \frac{L}{N} + \frac{1}{8} \cdot \frac{3L}{N} = \frac{4}{8} \cdot \frac{L}{N} \]

b. Corroboration of the last agent:

\[ [A \cdot x + u]_N \implies \frac{1}{4} \cdot x_{N-1} + \frac{1}{4} \cdot x_N + \frac{L}{2} = x_N \]

\[ \frac{1}{4} \cdot \bar{x}_{N-1} + \frac{1}{4} \cdot \bar{x}_N + \frac{L}{2} = \bar{x}_N \]

\[ \frac{1}{4} \cdot \frac{(2 \cdot (N-1) - 1) \cdot L}{2N} + \frac{1}{4} \cdot \frac{(2N-1) \cdot L}{2N} + \frac{L}{2} = \frac{(2N-1) \cdot L}{2N} \]

\[ \frac{(2N-3) \cdot L}{8N} + \frac{(2N-1) \cdot L}{8N} + \frac{4 \cdot L \cdot N}{8N} = \frac{(2N-1) \cdot 4L}{8N} \]

\[ 2N - 3 + 2N - 1 + 4N = 8N - 4 \]
c. Corroboration of the rest of the agents:

\[
[A \cdot x + u]_i = x_i
\]

\[
\frac{1}{4} \cdot \bar{x}_{i-1} + \frac{1}{2} \cdot \bar{x}_i + \frac{1}{4} \cdot \bar{x}_{i+1} = \bar{x}_i
\]

\[
\frac{1}{2} \cdot \bar{x}_{i-1} + \frac{1}{4} \cdot \bar{x}_{i+1} = \frac{1}{2} \cdot \bar{x}_i
\]

\[
\frac{1}{2} \cdot \frac{(2 \cdot (i - 1) - 1) \cdot L}{2N} + \frac{1}{2} \cdot \frac{(2 \cdot (i + 1) - 1) \cdot L}{2N} = \frac{(2i - 1) \cdot L}{2N}
\]

\[
\frac{1}{2} \cdot \frac{(2i - 2 - 1) \cdot L + (2i + 1 - 1) \cdot L}{2N} = \frac{(2i - 2) \cdot L}{2N}
\]

\[
\frac{1}{2} \cdot \frac{(4i - 2) \cdot L}{2N} = \frac{(2i - 1) L}{2N}
\]

When the three hypothesis are validated, the system converges and the proof of proposal 1 has finished.

5.1 Stability of matrix A

In this section we make a proof of the stability of the matrix A showed in the equation (5.0.10). The matrix A has a dimension \(N \times N\) where \(N\) is the number of agents in the system.

Firstly, we want to show that \(x_i = f(x_1)\) using the system \(A \cdot x = x\) where \(x\) is the vector of the positions of each agent at the end of the iteration process.

\[
\begin{pmatrix}
\frac{1}{4} & \frac{1}{4} & 0 \\
\frac{1}{4} & \frac{1}{2} & \frac{1}{4} \\
0 & \frac{1}{4} & \frac{1}{4}
\end{pmatrix}
\begin{pmatrix}
x_1 \\
x_2 \\
x_3
\end{pmatrix} = 
\begin{pmatrix}
x_1 \\
x_2 \\
x_3
\end{pmatrix}
\]

To continue, we are going to solve the system.

Starting with the first equation,

\[
\frac{1}{4}x_1 + \frac{1}{2}x_2 = x_1
\]

\[
x_2 = 3x_1
\]

Continuing with the second equation,
Using the equation (5.1.1) we can say:
\[ x_3 = 5x_1 \] (5.1.2)

Finishing with the last equation,
\[ \frac{1}{4}x_2 + \frac{1}{4}x_3 = x_3 \]

Using the equations (5.1.1) and (5.1.2) we arrive to the last equation,
\[ \frac{1}{4}x_2 = \frac{3}{4}x_3 \]

Finally, we can conclude that the only possibility to solve the system is that \( x_1, x_2 \) and \( x_3 \) are zero so we can ensure that the matrix is stable and the system converges to zero.

In order to make a good proof, we are going to do the same process but with a larger dimensional matrix.

\[
\begin{pmatrix}
1 & 1 & 0 & 0 \\
1 & 2 & 1 & 0 \\
0 & 1 & 2 & 1 \\
0 & 0 & 1 & 1 \\
\end{pmatrix}
\begin{pmatrix}
x_1 \\
x_2 \\
x_3 \\
x_4 \\
\end{pmatrix}
=
\begin{pmatrix}
4x_1 \\
4x_2 \\
4x_3 \\
4x_4 \\
\end{pmatrix}
\]

First equation,
\[ x_1 + x_2 = 4x_1 \]
\[ x_2 = 3x_1 \] (5.1.3)

Second equation,
\[ x_1 + 2x_2 + x_3 = 4x_2 \]

Using the equation (5.1.3), we deduce:
\[ x_3 = 5x_1 \] (5.1.4)

Third equation,
\[ x_2 + 2x_3 + x_4 = 4x_3 \]
Using the equations (5.1.3) and (5.1.4), we deduce:

\[ x_4 = 7x_1 \]  

(5.1.5)

Fourth and last equation,

\[ x_3 + x_4 = 4x_4 \]

\[ x_3 = 4x_4 \]

Using the equations (5.1.4) and (5.1.5),

\[ 5x_1 = 21x_1 \]  

(5.1.6)

\[ x_1 = 0 \]

Once we have studied this two specific cases, we want to ensure that \( x_i = (2i - 1)x_1 \) and we are going to study a generic case.

First of all, we are going to study the second agent. \((i = 2)\) but we have to show the equations that will be used in the proof.

The first one refers to the agent \( i - 1 \).

\[ x_{i-1} = (2(i - 1) - 1)x_1 \]  

(5.1.7)

\[ x_{i-1} = (2i - 3)x_1 \]

The second one refers to the agent \( i + 1 \).

\[ x_{i+1} = (2(i - 1) - 1)x_1 \]  

(5.1.8)

\[ x_{i+1} = (2i - 1)x_1 \]

Now, we are ready to make the corroboration explained before.

\[ x_{i-1} + 2x_i + x_{i+1} = 4x_i \]

\[ x_{i+1} = 2x_i - x_{i-1} \]

Applying the equations (5.1.8) and (5.1.7),

\[ (2i + 1)x_1 = 2(2i - 1)x_1 - (2i - 3)x_1 \]

\[ (2i + 1)x_1 = (4i - 4 - 2i + 3)x_1 \]
Before extracting erroneous conclusions, we have to make the corroboration of the last agent, where $i = N$.

\[ x_{N-1} + x_N = 4x_N \]
\[ x_{N-1} = 3x_N \]

Applying the equations (5.1.7) where $i = N$

\[ (2(N - 1))x_1 = 3(2N - 1)x_1 \]
\[ (6N - 3 - 2N + 2 + 1)x_1 = 0 \]
\[ x_1 = 0 \]

As we can see, the hypothesis made at the beginning of this section was right. If $x_1$ is zero, we can say $x_i = 0$ because of the dependence on $x_1$ of $x_i$. 

16
6 Alternative way to proof convergence

Before continuing to the 2D-dimensional case, we wanted to show how is possible to express our system without a constant vector \((u)\) and keep studying the method to calculate the minimum number of iterations.

6.1 Proof of convergence

Firstly, we would like to express the final position of each agent in the algorithm as an approach to the theoretical position and it will be defined by the difference between them.

\[
\tilde{x}_1 = x_1 - \frac{L}{2N}, \\
\tilde{x}_i = x_i - \frac{(2i-1) \cdot L}{2N}.
\]

(a) The error will tend to zero in an infinite time, so we will try to deduce the next system by the equations proved before.

\[
\tilde{x}(k+1) = A \cdot \tilde{x}(k), \quad \tilde{x}(k) \to 0
\]

(b) As it is known, the position in the next iteration of one agent will be defined by the current position of the agent and one agent of each neighbours.

i. The first corroboration is carried out with the first and second agents. We proceed from the equation of the first agent of the previous matrix system

\[
x_1(k+1) = \frac{1}{4} \cdot x_1(k) + \frac{1}{4} x_2(k)
\]

In order to get \(\tilde{x}_1(k) : \frac{L}{2N}\) is subtracted on both sides

\[
x_1(k+1) - \frac{L}{2N} = \frac{1}{4} \cdot x_1(k) + \frac{1}{4} x_2(k) - \frac{L}{2N},
\]

\[
\frac{L}{2N} = \frac{1}{4} \cdot \frac{L}{2N} + \frac{1}{4} \cdot 3L
\]

The previous equation permits expressing the right side of the equation as a function of \(\tilde{x}_1(k)\) and \(\tilde{x}_2(k)\)

\[
x_1(k+1) - \frac{L}{2N} = \frac{1}{4} \cdot (x_1(k) - \frac{L}{2N}) + \frac{1}{4} \cdot (x_2(k) - \frac{3L}{2N}),
\]

\[
\tilde{x}_1(k+1) = \frac{1}{4} \cdot \tilde{x}_1(k) + \frac{1}{4} \cdot \tilde{x}_2(k).
\]
Finally, it is possible to obtain the new system without any constant vector

\[ \tilde{x}(k + 1) = A \cdot \tilde{x}(k) \]

\[ \tilde{x}(k) \to 0 \]

ii. The second corroboration is carried out with the last two agents. We proceed from the equation of the last agent of the matrix system

\[ x_N(k + 1) = \frac{1}{4} x_N(k) + \frac{1}{2} x_{N-1}(k) + \frac{L}{2}. \]

As in the previous case, \( \frac{(2N-1)L}{2N} \) is subtracted on both sides

\[ x_N(k + 1) - \frac{(2N-1)L}{2N} = \frac{1}{4} x_N(t) + \frac{1}{4} x_{N-1}(k) + \frac{L}{2} \cdot \frac{(2N-1)L}{2N} \]

To arrive to the final equation we can express the subtraction as a sum of two factors

\[ \frac{(2N-1)L}{2N} - \frac{1}{2} \cdot \frac{(2N-1)L}{2N} = \frac{1}{4} \cdot \frac{3(2N-1)L}{2N}, \]

To find the coefficient of the term \( x_{N-1}(k) \), we can join the following factors

\[ \frac{3(2N-1)L}{2N} - \frac{L}{2} = \frac{(2N-3)L}{2N}, \]

\[ x_N(k + 1) - \frac{(2N-1)L}{2N} = \frac{1}{4} \left( x_N(k) - \frac{(2N-1)L}{2N} \right) + \frac{1}{4} \left( x_{N-1}(k) - \frac{(2N-1)L}{2N} \right), \]

\[ \tilde{x}_N(k + 1) = \frac{1}{4} \cdot \tilde{x}_N(k) + \frac{1}{4} \cdot \tilde{x}_{N-1}(k), \]

\[ \tilde{x}(k + 1) = A \cdot \tilde{x}(k). \]

iii. The last corroborations includes the rest of the agents. We proceed from the equation of a generic agent of the matrix system

\[ x_i(k + 1) = \frac{1}{2} x_i(k) + \frac{1}{4} x_{i-1}(k) + \frac{1}{4} x_{i+1}(k). \]

As in the two previous cases, \( \frac{(2i-1)L}{2N} \) is subtracted on both sides
\[ x_i(k + 1) = \frac{(2i-1)L}{2N} = \frac{1}{2} \cdot x_i(k) + \frac{1}{4} \cdot x_{i-1}(k) + \frac{1}{4} \cdot x_{i+1}(k) - \frac{(2i-1)L}{2N}. \]

To arrive to the final equation we can express the subtraction as a sum of two factors

\[ \frac{(2i-1)L}{2N} = \frac{1}{4} \cdot \left( \frac{2(i-1)-1}{2N} \right) + \frac{1}{2} \cdot \left( \frac{2i-1}{2N} \right) + \frac{1}{4} \cdot \left( \frac{2(i+1)-1}{2N} \right). \]  \hspace{1cm} (6.1.1)

In conclusion, the final expression is

\[ x_i(k + 1) = \frac{1}{2} \cdot x_i(k) + \frac{1}{4} \cdot x_{i-1}(k) + \frac{1}{4} \cdot x_{i+1}(k), \]

\[ \tilde{x}(k + 1) = A \cdot \tilde{x}(k). \]

To conclude, we have revalidated the proposal 1 with the purpose of achieving a valid and convergent theorem.

### 6.2 Convergence rate

Then, to perform a comprehensive study of the proposal, we will show how to calculate the minimum number of iterations from the matrix system. The best way to find \( k \) (number of iterations) is to restrict the error (difference between the relative position of the same agent between the temporary and the final theoretical position) to a predefined maximum value:

\[ ||\tilde{x}(k + 1)|| \leq ||A|| \cdot ||\tilde{x}(k)||, \]

For \( k = 0 \rightarrow ||\tilde{x}(1)|| \leq ||A|| \cdot ||\tilde{x}(0)||, \)

For \( k = 1 \rightarrow ||\tilde{x}(2)|| \leq ||A|| \cdot ||\tilde{x}(1)||. \)

Expressing \( \tilde{x}(2) \) in function of \( \tilde{x}(0) \)

\[ ||\tilde{x}(2)|| \leq ||A||^{2} \cdot ||\tilde{x}(0)||. \]

Deduction of the generic expression for \( \tilde{x}(t) \)

\[ ||\tilde{x}(k)|| \leq ||A||^{k} \cdot ||\tilde{x}(0)|| \leq \varepsilon. \]
Now, it is required to isolate $t$ from the inequality

$$
\|A\|^k \cdot \|\hat{x}(0)\| \leq \varepsilon,
\|A\|^k \leq \frac{\varepsilon}{\|\hat{x}(0)\|}.
$$

Applying logarithms and exploiting their properties we found that

$$
\log(\|A\|^k) \leq \log \left( \frac{\varepsilon}{\|\hat{x}(0)\|} \right),
$$

$$
k \cdot \log(\|A\|) \leq \log \left( \frac{\varepsilon}{\|\hat{x}(0)\|} \right),
$$

$$
k \geq \frac{\log \left( \frac{\varepsilon}{\|\hat{x}(0)\|} \right)}{\log(\|A\|)}.
$$

Recalling that the norm of $A$ is the largest eigenvalue of the matrix.

Finally, $k$ is the minimum number of iterations needed to converge with an error smaller than the value prefixed.
7 Voronoi partitioning of an area

In this section we review the problem of monitoring a two-dimensional environment of finite dimensions with \( N \) agents, where \( N \) is a finite number. Each agent has associated a defined area and according to this, it is possible to have two different situations. One happens when all the agents are synchronised and they communicate among them at the same time in order to cover the same area. The other one happens when the communication is only between one pair of agents, so the algorithm is asynchronous.

The agents are labelled 1 through \( N \), and for the sake of simplicity, we assume the following characteristics:

a. Each agent has the same memory,

b. Each agent will control a part of the environment of the same area.

7.1 Synchronous case

In this case, as it is a synchronous case, all the agents are going to move and update their position at the same rate. Therefore, in every step of the algorithm every agent receives and send information to its neighbours and updates its position considering them.

We assume the following hypothesis:

i. All the agents are synchronized, each one sends and receive information at the same time,

ii. All the agents are communicating among them.

In our analysis, we assume that there is no overlapping either blank space between the areas controlled by the agents. The space to be covered is delimited by four straight lines with the following equations:

\[
\begin{align*}
  y &= 0 \\
  x &= 0 \\
  y &= h \\
  x &= w
\end{align*}
\] (7.1.1)

To develop the algorithm, we have considered two different ways. The first one is based in the distance from each pixel to all the agents and associate this pixel to the nearest agent. The other one is based on the lines defined by the Voronoi partitions and the space covered by each agent. This two algorithms are explained in the next subsections.
Algorithm 5 Synchronous Symmetric Case 1

Require: $N, w, h, d$

1: while finished = false do
2:   for $x = 1, \ldots, w$ do
3:     for $y = 1, \ldots, h$ do
4:       For each pixel, perform the equations (7.1.2) to calculate which is the nearest agent
       and assign it to this agent
5:       end for
6:     end for
7:   for $i = 1, \ldots, N$ do
8:     for $x = 1, \ldots, w$ do
9:       for $y = 1, \ldots, h$ do
10:      For each agent perform the equations (7.1.3) to compute its new centroid, considering the area needed to cover.
11:    end for
12:   end for
13: end for
14: for $i = 1, \ldots, N$ do
15:   if $a(i) = w \times h/N$ then
16:     finished = true
17:   end if
18: end for
19: end while

7.1.1 Algorithm 1

First, the position of the agents in the area is selected randomly, where the $x$ component has to be in $w$ and the $y$ component in $h$. Then, as the theory of Voronoi partitions said, we have to compute the distance from every pixel in the area to all of the agents, and this pixel will be associated to the agent whose distance to him is lower. So, the space will be split up in different regions.

The equations to compute the distance to each agent are the following, where $p_i$ is a vector with the position of each agent, $d_i$ is the distance to each agent and $[x,y]$ is the pixel which we are computing the distance to each agent:

\[
[d_1(1), d_1(2)] = [p_i(1), p_i(2)] - [x, y]
\]
\[
d_i = \sqrt{d_1(1)^2 + d_1(2)^2}
\]  

(7.1.2)

To continue with the algorithm, it is computed the centroid of each area and it becomes the new position of the agent. The next equations refer to the computation of the centroid of each area.
Now, it will be necessary to recalculate the distance from each pixel to each agent and the next centroid. This process it is repeated until the total area is distributed homogeneously.

\[
\begin{align*}
    c_x &= \frac{\sum x}{a_i} \\
    c_y &= \frac{\sum y}{a_i} \\
    c &= [c_x, c_y]
\end{align*}
\]  

(7.1.3)

7.1.2 Algorithm 2

**Algorithm 6 Synchronous Symmetric Case 2**

Require: \( N, w, h, p, M_i, A_j \)

1: while \( \text{step} < \text{NumberOfSteps} \) do
2:     for \( i = 1, ..., N \) do
3:         for \( j = 1, ..., N \) do
4:             if \( j \neq i \) then
5:                 Perform the equations (7.1.4) to obtain the line of the agent \( i \) and the agent \( j \).
6:             end if
7:         end for
8:     end for
9:     for \( i = 1, ..., N \) do
10:        for \( j = 1, ..., N \) do
11:            if \( \text{The agent is on one side of the line} \) then
12:                Each pixel that belongs to the same side of the agent is a 1 on the matrix \( A_j \)
13:            end if
14:        end for
15:    end for
16:    for \( i = 1, ..., N \) do
17:        for \( x = 1, ..., \text{width} \) do
18:            for \( y = 1, ..., \text{height} \) do
19:                For each agent perform the equations (7.1.3) to compute its new centroid, considering the area needed to cover.
20:            end for
21:        end for
22:    end for
23: end while

As in the other way, firstly, the position of the agents in the area is selected randomly, where the \( x \) component has to be in \( w \) and the \( y \) component in \( h \). Then, we compute all the middle lines between one agent and the other agents. Each line has to be defined by one point and one vector. In order to compute these lines there are executed the following equations:
The middle point between two agents $i$ and $j$ is the point which belongs to the line $l$. $v_l$ is the perpendicular vector which links the two centroids of the agents $i$ and $j$.

Once we have the lines for each agent with all the other agents, we have to define the area covered by each agent. For each line of each agent we are computing a matrix where each element represents a pixel of the area. This matrix has a 1 if the pixel belongs to the agent, otherwise there is a 0. To continue, it is computed an element-wise multiplication between all the matrix lines of each agent so we can obtain the area of each agent.

Then, to continue with the algorithm, it is computed the centroid of the area of each agent and it becomes the new position of the agent. The equations that refer to the computation of the centroid of each area are the same used in section 7.1.1, equation (7.1.3).

Where $x$ and $y$ are the pixels that belong to the agent $i$.

Finally, after having computed the new centroid, it is needed to repeat the matrix lines process.

### 7.2 Asynchronous case

In this section we are going to compute just the symmetric algorithm because of the complexity of the asymmetric one. The computation of the symmetric algorithm is based in the communication between one agent with all its neighbours, computing the centroid and calculating the new area of each agent by the Voronoi partitions.

We assume the following hypothesis:

i. In each step, only one agent is updating its position to its new centroid,

ii. In each step, one agent is communicating with all its neighbours.

In our analysis, we assume that there is no overlapping either blank space between the areas controlled by the agents, as we said in the synchronous case. The space to be covered is delimited by four straight lines with the equations explained in (7.1.1).

To develop this algorithm, we have considered only one way which is explained in the following lines.

In the asynchronous case, to compute the middle lines between one agent and the others it is necessary to execute the same equations as in the synchronous case, (7.1.4). So the difference begins in the computation of the centroid, where only is calculated the centroid of one agent that is selected randomly and then he updates its position by communicating with his neighbours, executing the equation (7.1.3).
Algorithm 7 Asynchronous Symmetric Case

Require: $N, w, h, p, M_i, A_j$

1: while $step < NumberOfSteps$ do
2:   $i = random(N)$
3:   for $j = 1, ..., N$ do
4:     if $j! = i$ then
5:       Perform the equations (7.1.4) to obtain the line of the agent $i$ and the agent $j$.
6:     end if
7:   end for
8:   for $i = 1, ..., N$ do
9:     for $j = 1, ..., N$ do
10:       if The agent is on one side of the line then
11:         Each pixel that belongs to the same side of the agent is a 1 on the matrix $A_j$
12:       end if
13:   end for
14:   end for
15:   for $x = 1, ..., w$ do
16:     for $y = 1, ..., h$ do
17:       For the agent $i$ perform the equations (7.1.3) to compute its new centroid, considering the area needed to cover.
18:     end for
19:   end for
20: end while

7.3 Density distribution

In the 2D case we have done two different simulations. A first simulation where we assume that the distribution of the agents is independent to the importance of the area, so each pixel is as important as the other ones. In this case at the end each agent will control the same space of area. The other simulation considers a density distribution, where some pixels are more important than the others. So the distribution of the agents among the area will be depending on this density function. In this case an agent can control an area bigger than other agents. The density can represent the temperature which if it is higher could mean a fire is occurring so you want to monitor the area, the same happens when you want to control a city where the density function represents the probability of danger.

The density function that we are using is a Gaussian distribution defined by different parameters that determinates its properties. The equation of the density function is the following one:

$$ f(x, y) = c \times exp(-\frac{((x-x_1)^2 + (y-y_1)^2)}{(2n^2)}) + 1; $$

Where $x_1$ and $y_1$ determinates the center of the peak, $c$ is the value of the peak and $n$ the variance of the function. The function determinates where have to stay the agents to control the area correctly, in order to have more agents where there is a higher possibility of danger.

In the figure 3 there is a 3D representation of the distribution to better understand the allocation of the agents. In order to verify the functionality of the density function, we have tried the Gaussian distribution varying the parameters so the agents have to allocate in different size areas. The simulation results are shown in the subsection 8.2.1.
Figure 3: Gaussian distribution: the density function that the agents follow in the algorithm.
8 Simulation results

After computing the algorithm in Matlab, we have simulated every case that we have explained and we want to show an example of the results in order to easily understand the algorithm and corroborate the convergence of the algorithm. Firstly, we have done an exhaust study about the One Dimensional case in order to verify the validity of the algorithm. Then, we have done an extrapolation to the Two Dimensional case in order to make a real situation of video-surveillance and giving an example of what would be the behaviour of the agents.

8.1 1-Dimensional Case

In this section, we are going to show the results of the different algorithms that we have implemented. The steps to follow are explained with more detail in section 3 but, to sum up, we expect to have a linear distribution of the agents along the line. Each of the following pictures is a snapshot in a different step so we can see the evolution of the agents from the beginning to the final configuration. Every algorithm is simulated with six agents that have a random initial position in the length of the line, which is equal to 20 m. To stop the algorithm, in each case there is a need of having a constraint that has to be verified. In the next paragraphs we explain with more detail how is analysed each algorithm, the initial conditions and what is expected to obtain after the simulation.

Synchronous

To sum up, in one step all the agents communicate among them and update their positions in the line to cover the line. Because of its speed of convergence, the number of steps that we have analysed is low, in this case we have simulated twenty steps. We hope for having an homogeneous distribution of the six agents along the line at the end of the algorithm. It has to stop when the maximum number of steps is reached. We can observe the simulation results in the Figure 4.

Asynchronous Symmetric

In this case, in each step only two agents are communicating and only two of the agents are going to update its position, the algorithm is going to stop when the error is less than 0.1 and the number of steps done in the algorithm depends on the initial position of the agents. We still have six agents that have to cover the line so the error that determines the end of the algorithm is the difference between the ideal position of the agent at the end of the algorithm and its position in a step. The simulation results are shown in Figure 5.

Asynchronous Asymmetric

In the asynchronous and asymmetric case, in each step only one agent updates its position by communicating with one of its neighbours. The algorithm is simulated with six agents and 100 steps, because the speed of convergence is lower than in the synchronous case. As it is said in the explanation of the algorithm, we will take into account two different methods, one where it is possible to find holes without covering and the other is where we can not find holes between two neighbouring agents but we look forward to have the same final configuration in the two methods. The simulation results are shown in Figures 6 and 7.
Figure 4: Synchronous case: Steps 1 (Top left), 3 (Top right), 5 (Bottom left), 20 (Bottom right). Homogeneous distribution of the agents in a line where the way of communication is synchronous.
Figure 5: Asynchronous Symmetric: Steps 1 (Top left), 7 (Top right), 15 (Medium left), 25 (Medium right), 30 (Bottom). Distribution of the agents in an asynchronous way where only two agents are communicating and updating their position.
8.1.1 Convergence rate

As we have seen in the proof of One Dimensional Coverage, the error during the algorithm is getting lower, tending to zero. In the following figures we can see three lines of convergence, one belongs to the Synchronous Case, another to the gossip Symmetric and the other to the Asymmetric.

Knowing that the slope of the line is directly related to the speed of convergence, we have made the logarithm to the function of the error in order to obtain a line. So, in the figure we can see that the fastest algorithm is the one which is synchronous because in each step all the agents communicate among them. To conclude, as it is seen in the figure 8 and 9, the error without a logarithm tends to zero and with the logarithm we can see the algorithm which converges faster to zero is the synchronous and the lowest the asynchronous asymmetric.

8.2 2-Dimensional Case

In this section, we are going to show the results of the different algorithms that we have implemented in a two dimensional environment. The steps to follow are explained with more detail in section 7. Each algorithm is simulated with six agents, equal to the 1D case, and a squared area of 30 cm each side. We have only implemented two cases, the synchronous and the asynchronous asymmetric. The asynchronous and symmetric has not been implemented in this project because of the complexity of the algorithm. In both algorithms we have forced the initial conditions in order to better see the change of the position of the agents during the execution and the only limitation that is different in each algorithm is the number of steps. The following pictures are snapshots in different steps of the algorithm to better understand what is happening and what is expected to have.

Synchronous

As it is said in the previous section, the synchronous case is where all the agents communicate between them so in each steps all of them will update their position. As the communication is very fast and the convergence is reached early, the number of steps simulated is 20. At the beginning the agents are located in a small range of area concentrated in the right corner and we hope for having an homogeneous distribution of the six agents in the area at the end of the algorithm. The simulation results are shown in the Figures 10 and 11 with different styles of representation.

Asynchronous Symmetric

The type of communication is the same as in the 1D case, where only two agents out of six are going to change its position in each step, the speed to converge is lower so the number of steps is 50. When the final positions of each agent is reached before achieving the maximum number of steps, the simulation will continue to the end without updating the position of any agent. At the beginning the agents are located in the right corner of the area and, because there is no density function, the distribution is homogeneous. The simulation results are shown in Figures 12 and 13.
Figure 6: Asynchronous Asymmetric with holes: Steps 1 (Top left), 15 (Top right), 25 (Medium left), 35 (Medium right), 150 (Bottom). Distribution of the agents in an asynchronous way where only one agent is communicating with one of its neighbours and updating its position. During the algorithm is possible to have a space not covered.
Figure 7: Asynchronous Asymmetric without holes: Steps 1 (Top left), 7 (Top right), 15 (Medium left), 25 (Medium right), 100 (Bottom). Distribution of the agents in an asynchronous way where only one agent is communicating with one of its neighbours and updating its position. During the algorithm, in each step all the space in the line is covered.
8.2.1 Different density functions

We have seen the Gaussian function and its parameters but we still need to show the simulation results. We have done three extreme simulations, two of them have the peak in a corner and the other one has the peak in the center, so the area covered around the peak will be the most populated by the agents. With the finality to show a good simulation, we have forced the initial conditions so we can better see how the agents move directly to the peak. In the Figures 14, 15 and 16 we can see the final configuration of the agents.

8.2.2 Convergence rate

In the convergence rate of the Two Dimensional case, the error is calculated by the difference between the theoretical area of each agent with the final area of each agent in the simulation. The theoretical area is computed by dividing the total area in pixels by the number of agents so we obtain the number of pixels that theoretically has to belong to each agent. It is compared the error between two different algorithms, the synchronous and the asynchronous symmetric where we can see that the synchronous one is faster than the asynchronous, as we have seen in the One Dimensional case.

Furthermore, we observe that at the beginning the error decreases faster because the updated areas of each agent are much more distinct from the previous one, at the end the algorithm tries different manners to get closer to the optimal solution. In the Figure 17 we show the results of the simulations.

As in the 1D case, we have also plotted the logarithm of the error in order to see what is the speed of convergence of the two algorithms that we have implemented. We want to verify that the synchronous case converges faster than the asynchronous one due to the communication among all the agents. We can observe the results in the Figure 18.
9 Conclusions

We have presented a set distributed partitioning and coverage control algorithms which require a type of communication between pairs of robots and work in the 1D and 2D cases. The agents are spread to cover the optimal area. The algorithm involves a finite number of iterations of centering and partitioning method. We have proved that the 1D case converges and we show from extensively simulations that the 2D seems to converge as well.

We used two methods of communication called Synchronous and Asynchronous. We can conclude that the asynchronous is the one which resembles more to the reality, so we have started with the synchronous model in one dimension to finish making an asynchronous symmetric algorithm in a two dimensional environment.

Our vision is that this partitioning and coverage algorithms will form the foundation of a distributed task servicing setup for teams of mobile robots. The robots divide the area among them and move to contact their neighbours with the finality to improve the coverage. We also studied the case where the density function over the environment is not constant. In particular, if it is Gaussian, we can see that the agents concentrate themselves around the peak.

Our simulation results show an example of what would be the behaviour of the agents in a defined area so they communicate in order to cover it. The algorithm only requires sporadic improvements such as affording flexibility to the agents behaviours and capacities or giving more flexibility in the number of steps. In all the plots is shown the potential of gossip communication in distributed coordination algorithms. We can conclude that our algorithms could be successfully applied in many practical situations, where distributed and minimal communication models are required.
Figure 8: Plot of the error

Figure 9: Logarithm of the plot of the error
Figure 10: Synchronous Voronoi Partitions: Steps 1 (Top left), 3 (Top right), 5 (Bottom left), 20 (Bottom right). Homogeneous distribution of the agents in the area with a synchronous way of communication. In this algorithm, all the agents communicate among them. The blue line represents the border of the Voronoi partitions and the blue dots are the position of the agents.
Homogeneous distribution of the agents in the area with a synchronous way of communication. In this algorithm, all the agents communicate among them. All the pixels with the same colour owns the same Voronoi partitions.
Figure 12: Asynchronous symmetric: Steps 1 (Top left), 7 (Top right), 12 (Medium left), 20 (Medium right), 50 (Bottom). Homogeneous distribution of the agents in the area with an asynchronous way of communication. In this algorithm, only two agents are communicating between them and updating their position. The blue line represents the border of the Voronoi partitions and the blue dots are the position of the agents.
Figure 13: Asynchronous symmetric: Steps 1 (Top left), 3 (Top right), 7 (Medium left), 20 (Medium right), 50 (Bottom). In this algorithm, only two agents are communicating between them and updating their position. All the pixels with the same colour owns the same Voronoi partitions and the black points represent the position of the agents.
Figure 14: Final configuration of the partition algorithm. Gaussian Parameters: $x_1=5$, $y_1=5$, $n=10$, $c=10000$.

Figure 15: Final configuration of the partition algorithm. Gaussian Parameters: $x_1=50$, $y_1=50$, $n=10$, $c=5000$.

Figure 16: Final configuration of the partition algorithm. Gaussian Parameters: $x_1=0$, $y_1=90$, $n=10$, $c=9000$. 
Figure 17: Convergence rate 2D. It is seen how the error of the positions of the agents tend to zero, where the error is calculated by the difference between the theoretical area of each agent with the final area of each agent in the simulation.

Figure 18: Logarithm of the convergence rate 2D. It is seen how the synchronous case converges to zero easily than the asynchronous one due to the communication among all the agents.