Distributed Robots for Gas Monitoring in Unknown Environments

Davide Dalla Stella
223727
davide.dallastella@studenti.unitn.it

Simone Luchetta
223716
simone.luchetta@studenti.unitn.it

Abstract

Nowadays robots are employed in many contexts and require algorithms that make them capable of satisfying many tasks autonomously. In this work a distributed environment is implemented.
A multitude of robots cooperate to explore an area and analyze the amount of CO_2 present in the air, the final output will thus be a topography made up of the union of the various areas explored by the single agents on which the overall distribution of the air is applied. Overall, the payload of the agents will be composed of the following: a LiDAR sensor, an air quality measuring device and an ultra-wideband antenna. The application of these robots can be varied, from simple analysis of air quality, to the search for gas leaks in a duct to even finding the safest path for an operator to follow during a fire. This work was implemented through a matlab simulation. All the techniques used to achieve this goal will be herein presented.

1. Introduction

The ability of establishing a reliable cooperative control is essential for distributed robotic networks. Many studies show that it is possible to have accurate and robust localization information without using inertial navigation system (INS) or global positioning system (GPS) and by using solely measures based on the distances between each pair of nodes inside the network, even for large-scale scenarios as in [5].

Very often the technique employed in order to determine the location of each component in the network is the Multidimensional scaling (MDS). Altough the elements needed to perform the localization by means of MDS are only the inter-distances measures between the nodes, the usage of this technique is not trivial as the reconstruction of the information is subject to reflection, rotation and translation, as stated in [4].

One of the strengths of the MDS algorithm lies in the fact that its use in distributed systems and in many other fields has been studied extensively over time. In [2], a particular version of the algorithm has also been created that takes into consideration the non-direct connection between two nodes belonging to the same network.

Concerning this project, several agents performs the exploration of an unknown environment. As each agent moves around the map, both a scan of its proximity and air quality is acquired, using a LiDAR and CO_2 sensor, respectively. So, the LiDAR gives a field of vision that permits the agents to compute a pathway to unexplored safe points that are off obstacles. In the meantime, gathered air quality data are averaged and processed using a Kalman Filter.

Beyond this, each agent is equipped with an ultra-wideband (UWB) tracking system that allows metering techniques. More specifically, each robot computes the distance between itself and the others. Then, all so obtained measure are gathered together to build up a distance matrix. This matrix is used via the MultiDimensional Scaling (MDS) to find a common reference system for all agents within which they are able to move and locate themselves. Hence, the concept is that the spatial configuration of how agents are moving and scout the areas is reconstructed basing solely on the information coming from inter-distances of the robots, from which positions of the nodes in the network are estimated.

2. Problem Statement

The work is implemented through a matlab simulation. 4 rovers are positioned in different areas of a map (Fig. 1) that is unknown to them and they have to explore it while capturing air quality data (simulated like in Fig. 2). Each robot is equipped with a sensor that analyzes the amount of CO_2 in the air, a LiDAR sensor to scan the surrounding area and an ultra-wideband module used to calculate distances from other robots. The problem was divided into several stages:

1. Generation of the reference system: it is generated through the MDS method using a robot as anchor.

2. Data acquisition: for each robot use the lidar sensor and the CO_2 sensor to generate a local map with the CO_2 concentration.

3. Exploration: for each robot, compute the path in order to reach a new position, where a new scan will be retrieved.

4. Final map generation: put together the data captured by all the robots and generate a global map with the distribution of the CO_2 gas distribution.

In short, at the beginning a robot is chosen in an arbitrary way which will be the reference point for the creation of the reference system. To do so, the distance matrix from the other robots is computed thanks to the ultra wideband module and the reference system is obtained via the MDS. Once this is done, each robot is free to start exploring the map. It scans its starting point, thanks to which it obtains information on the surrounding area and saves the value of the concentration of CO_2 in the air with the associated position. Subsequently it begins to move so as to be able to make a new acquisition of the map to merge with the one already obtained and enlarge its dataset on CO_2. With each movement performed, a robot takes care of sharing its updated position to the other agents so that they always know where they are in real time. After the agents have finished exploring, they will send their data to a central agent who will simply merge all the information into a single map on which it will apply the overall CO_2 distribution.

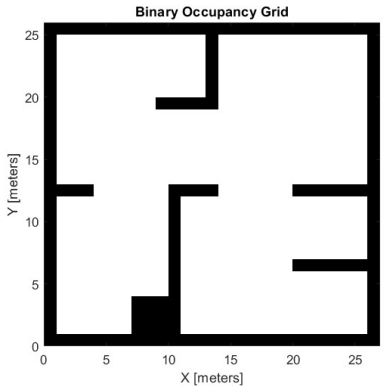

Figure 1. Map where the agents are located

3. Reference System Generation

To compute the reference sytem the MDS technique is used. This method is widely used for visualizing a set of objects in an n-dimensional space reproducing as much as possible the distances between the original points. As input this method takes an array of spacings. Each row of the matrix contains the proximity data of one agent from the others and therefore the matrix is displayed as triangular where the diagonal is composed of 0 and the upper triangular part mirrors the lower one with all the values ≥ 0 (like in Tab. 1).

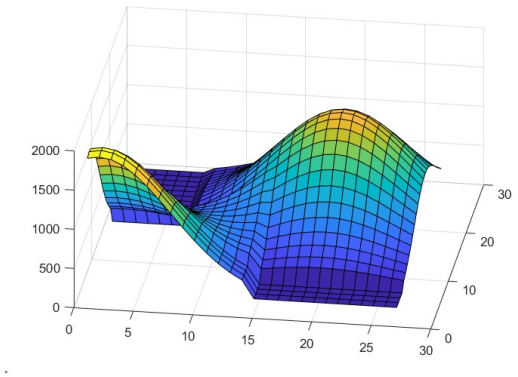

Figure 2. CO_2 Distribution

0	4	16.4	16.7
4	0	14.3	21.6
16.4	14.3	0	18.7
19.7	21.6	18.7	0

Table 1. Example of Interdistance Matrix, where each row is filled as in Eq. (1)

So, the MDS algorithm takes as input a Dissimilarity Matrix, that expresses how much two objects are dissimilar along one quality and finds a set of coordinates such that the distance between each couple of objects is proportional to the value of dissimilarity according to the loss function in Eq. (3) that is often referred to as *strain* or *stress*: it is basically a measure that indicates how much a high dimensional representation can be compressed into a lower dimensional solution. For the scenario of this project, objects are represented by the rovers and the dissimilarity matrix is built up with the inter-node distances and it is referred to as $\mathbf{D}$.

The true distance between the anchor node at $X_0 = [x, y]^T$ and one of its neighbours $X_i = [x_i, y_i]^T$, for $i = 1, 2, ..., n$ where n is the number of rovers moving inside the map is given by Eq. (1).

$$d_i = \sqrt{(X_0 - X_i)^T (X_0 - X_i)} \tag{1}$$

Though, the range measurement r_i between the anchor x_0 and one neighbour x_i is often corrupted by noise q_i, and this yields Eq. (2).

$$r_i = d_i + q_i,\ i = 1, 2, ..., n \tag{2}$$

Where q_i is modeled to be a gaussian random noise and depends on the parameters mu_r and $sigma_r$ in the Matlab code. Overall, this perturbation affects the values reported in Tab. 1 as shown in Tab. 2.

0	4.16	16.50	19.81
4.16	0	14.49	21.65
16.50	14.49	0	18.96
19.81	21.65	18.96	0

Table 2. Interdistance Matrix being affected by noise in the ranging measurements as in Eq. (2)

The loss function of the MDS algorithm that needs to be minimized is the following:

$$Strain_D(x_1, \ldots, x_n) = \left(\frac{\sum_{i,j} (b_{ij} - x_i^T x_j)^2}{\sum_{i,j} b_{ij}^2} \right)^{1/2} \quad (3)$$

Where b_{ij} are the elements of the matrix $\mathbf{B}$ that is introduced further below, and x_i, x_j are the vectors of n features containing the row distances (spatial information) for robot X_i and X_j.

For a basic understanding of how MDS works, we refer to [6]. Instead, the steps for solving the classical MDS algorithm are reported in [3] and are briefly summarized in the following procedure:

- Get the squared proximity matrix $\mathbf{D^{(2)}} = [d_{ij}^2]$, that is composed of distances between robot X_i and X_j.
- Compute the center matrix $\mathbf{J} = \mathbf{I}_{n+1} - \frac{1}{n+1}\mathbf{1}_{n+1}\mathbf{1}_{n+1}^T$. Using this notation, $\mathbf{I}_{n+1}$ represents the identity matrix with dimensions $(n+1) \times (n+1)$, while $\mathbf{1}_{n+1}$ represents the column vector of all ones having dimensions $(n+1) \times 1$
- Obtain the scalar product matrix $\mathbf{B}$ from the Inter-Distance matrix $\mathbf{D}$ via a double centering operation: $\mathbf{B} = -\frac{1}{2}\mathbf{JDJ}^T$.
- Decompose $\mathbf{B}$ and obtain its eigenvalues λi and eigenvectors e_i. Eigenvalues and eigenvectors need to be sorted from largest to smallest. It is then possible to select the dimensions for the output by truncating the first $M \leq N$ eigenvalues and eigenvectors.

Rovers proceed to share their information about the distances between each other, and build the matrix $\mathbf{D}$. Passing this matrix to the MDS algorithm leads to acquire an initial system configuration that is depicted in **??**. Then, a rover is picked to perform a series of actions:

- Step 1: compute a first MDS estimate while in position P_0. Center the obtained configuration around the position of the chosen rover, as in the example in Tab. 3.
- Step 2: move/push forward a determined amount of distance d in a direction that is cleared off obstacles, to reach position P_1. After this, get an update of $\mathbf{D}$ and call the MDS algorithm in order to get a second estimate of the configuration. Then again, center the system around P_1.
- Step 3: make an orthogonal turn, drive on for the same distance d once again and reach position P_2. Then, as before, call the MDS algorithm with an updated matrix $\mathbf{D}$ to get the third estimate, and center it around P_2.
- Step 4: search for mismatches between configurations of MDSs taken in P_0, P_1 and P_2. This is achieved by solving an optimization problem, which for this project was carried out by means of a genetic algorithm - more on that later on. Next, reflections are executed by applying a rotation as in Eq. (4) accounting for corruptions detected. In this case, θ_i corresponds to the rotation angle found by the genetic algorithm, which represents the rotation displacement between the system configurations at previous steps, i.e. when the rover was in P_0, P_1 and P_2.

System configuration captured at P_0:

x	6.13	7.94	-0.47	-13.60
y	4.97	1.61	-10.13	3.53

After centering on the rover (ID = 1):

x	0.00	-1.81	6.60	19.74
y	0.00	-3.36	-15.10	-1.43

Table 3. Performing step 1: MDS configuration taken at P_0 before and after being centered on the chosen rover

$$\text{Rotation Matrix} \begin{bmatrix} cos(\theta_i) & -sin(\theta_i) \\ sin(\theta_i) & cos(\theta_i) \end{bmatrix} \quad (4)$$

The aim of this process is to find a reliable common reference system that can be shared amongst all rovers. The main problem is the need to find the rotational translation from a reference system taken at position P_0, from the one taken at position P_1. If there are flips or rotations, these must be corrected, otherwise the data coming from the other robots when composing the complete map could be misreported or mixed. Then again, the same concept is applied between the pairs of systems taken at P_1 and P_2 respectively.

The chosen method is to solve this optimization problem through the help of a genetic algorithm. Here, the general idea is to have various iterations in which a population of solutions evolves towards an optimum.

Given a pair of MDS configurations, namely MDS_i and MDS_j that are centered respectively at positions P_i and P_j with $i < j$, we get the matrix R by following the series of operations in Eq. (5), Eq. (6) and Eq. (7):

$$H = MDS_i MDS_j^T \tag{5}$$

$$[U, S, V] = svd(H) \tag{6}$$

$$R = VU^T \tag{7}$$

Where $svd(H)$ represents the singular value decomposition on matrix H, such that $H = USV^T$, where S is a diagonal matrix of the same dimension as H and with non-negative diagonal elements in decreasing order, and U and V are unitary matrices.

If $det(R)$ is negative, it means that there has been a reflection between the system MDS_i and MDS_j, so it must be accounted later on.

Proceeding, the generative algorithm finds a local minimum subjected to the fitness function in Eq. (8):

$$fitness\_error = \sqrt{(MDS_i - rot_k(MDSj))^2} \tag{8}$$

Where rot_k is an affine transformation composed of mirroring, rotation and translation of the system MDS_j by the vector of input parameters $\lambda = [\lambda_1, \ldots, \lambda_i, \ldots, \lambda_5]$ of the algorithm. When the $fitness\_error$ is within an acceptable threshold, the input parameters are returned and used for matching the configuration of MDS_j to be roughly the same as the one of MDS_i as in Eq. (9):

$$MDS_i = \begin{bmatrix} cos(\lambda_1) & -sin(\lambda_1) \\ sin(\lambda_1) & cos(\lambda_1) \end{bmatrix} \begin{bmatrix} \lambda_4 & 0 \\ 0 & \lambda_5 \end{bmatrix} \times MDS_j + \begin{bmatrix} \lambda_2 \\ \lambda_3 \end{bmatrix} \tag{9}$$

Where λ_1 represents the displacement angle, $\lambda_{2,3}$ account for translation and finally $\lambda_{4,5}$ indicate if there has been any mirroring between configuration MDS_i and MDS_j.

The process iterates until the tuple of systems collected at P_0, P_1 and P_2 have no reflections around x, y axis or mirroring. This yields a system in which each robot refers to the robot that performed the disambiguation maneuvers.

4. Data Acquisition

Each rover comes equipped with a LiDAR (light detection and ranging) and a CO_2 sensor. The LiDAR sensor is a laser scanner for a 2-D plane with distances (up to a maximum detection range, specified as a positive scalar in meters) measured from the sensor to obstacles in the environment at specific angles. The sensor cannot detect roads and actors beyond the specified range.

Each robot acquires the information and saves them with their associated position to then move to a new area and make a new acquisition. The environment information are captured by the LiDAR scan while the CO_2 data by the gas sensor. The CO_2 data are filtered using an Kalman filter.

4.1. CO_2 Data Correction

Whenever a new gas measure is acquired, the Kalman filter smoothing is applied in order to get an estimate which is as close as possible to the true value. The aim is to reconstruct the distribution of gas from the underlying true source depicted in Fig. 7.
The followed steps are discussed in this section and depicted in Fig. 3.

4.1.1 Kalman filter

The Kalman filter is a type of optimal state filter used to estimate the state of a dynamical system based on uncertain sensory measurements. This filter is based on a dynamic state model that describes how the system changes over time and on an observation model that describes how the sensor measurements depend on the state of the system.
There are two kind of noises involved during the estimation of the system state, one of them being the *Measurement Noise* and the other being the *Process Noise*. The first regards the errors included in the measurements, while the latter is about the dynamic model error.

To understand the principle of working and explain the Kalman filter algorithm, first the same notation as in [1] is adopted:

- x: is the true value.
- z_n: is the measured value at time n.
- $\hat{x}_{n,n}$: is the estimate of x at time n (the estimate is made after taking the measurement z_n).
- $\hat{x}_{n+1,n}$: is the estimate of the future state $(n+1)$ of x. The estimate is made at the time n. In other words, $\hat{x}_{n+1,n}$ is a predicted state or extrapolated state.
- $\hat{x}_{n-1,n-1}$: is the estimate of x at time $n-1$ (the estimate is made after taking the measurement z_{n-1}).
- $\hat{x}_{n,n-1}$: is a prior prediction - the estimate of the state at time n. The estimate is made at the time $n-1$.

The Kalman filter is particularly useful for estimating the state of a system $\hat{\mathbf{x}}_{\mathbf{n,n}}$ when sensor measurements are subject to noise or uncertainty.

In Fig. 4 and Fig. 5 it is possible to see the difference between the estimates of the algorithm (blue line) and the true values of CO_2 (green line) when the robot is in a certain position around the map. The difference between the

filtered values and the true value is called *estimate error*, and it becomes lower as more and more measurements are acquired converging to the true value (green line). Though, it is never possible to know the estimate error, but it is possible to estimate the *state uncertainty*, which will be referred to as **p**.

Instead, the *measurement errors* are a random variable which is equal to the differences between the noisy measurements (in red) and the true values. The variance of the random variable is denoted by the symbol σ^2, while the *measurement uncertainty* is indicated as $\mathbf{r}$. The latter is often provided by the manufacturer of the equipment or can be derived empirically by a calibration procedure.

The Kalman filter algorithm is depicted in Fig. 3 and can be summarized as follows:

1. Initialization: this step is carried out only once, at it is needed in order to set the system's initial state $\hat{x}_{0,0}$ and the system's initial state uncertainty $p_{0,0}$.

2. Measurement: CO_2 scans are collected, providing the state of the system z_n with its associated measurement uncertainty r_n.

3. Update: Eq. (10) is used in order to get an estimate of the current state $\hat{x}_{n,n}$, using the Kalman gain (Fig. 6) computed as in Eq. (11). Of course, a low measurement uncertainty r_n would yield a Kalman gain close to 1, so that the estimate $\hat{x}_{n,n}$ would be close to the measurement z_n. Instead, an high measurement uncertainty r_n would give a Kalman gain close to 0, which in turn would favour the prior estimate $\hat{x}_{n,n-1}$.

4. Prediction: the filter uses the dynamic state model to estimate the future state of the system. The prediction outputs are namely the prior predicted state estimate $\hat{x}_{n,n-1}$ and the prior state uncertainty $p_{n,n-1}$ that will replace the variables $\hat{x}_{0,0}$ and $p_{0,0}$ used at initialization. These variables are kept updated for future iterations.

$$\hat{x}_{n,n} = (1 - K_n)\hat{x}_{n,n-1} + K_n z_n \tag{10}$$

$$K_n = \frac{p_{n,n-1}}{p_{n,n-1} + r_n} \tag{11}$$

More in detail, in the matlab simulation roughly $k = 100$ CO_2 scans are acquired when the robot is standing still at a current position, prior to moving to the next one, and each measure is perturbed by a random *Measurement Noise* as in Eq. (12).

$$z_n = x_{x,y} \pm \psi \tag{12}$$

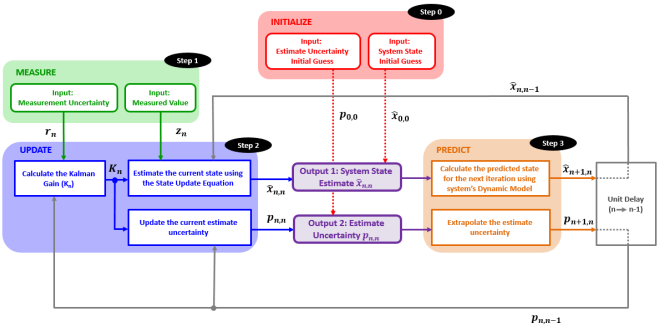

Figure 3. Schematic description of the Kalman filter [1].

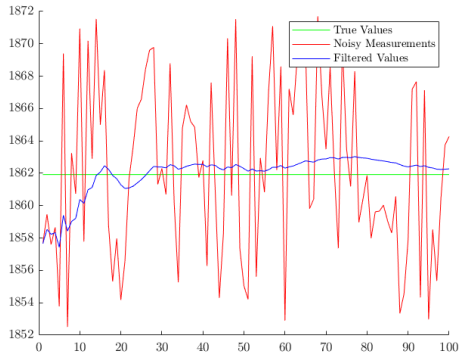

Figure 4. Detail of the noisy sensor measurements (in red) being processed by the Kalman algorithm (result available in blue).

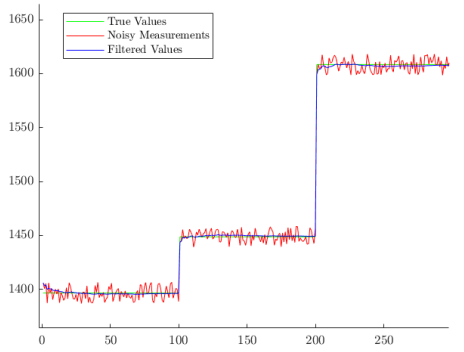

Figure 5. Noisy sensor measurements filtered by Kalman algorithm when the robot wanders around the map.

Where ψ represents the random value taken from a continuous uniform distribution perturbing the true measurement $x_{x,y}$ at position x, y.

4.2. Scan Area

As regards the simulation part of the acquisition of the LiDAR scans, matlab needs two things as input: one is the

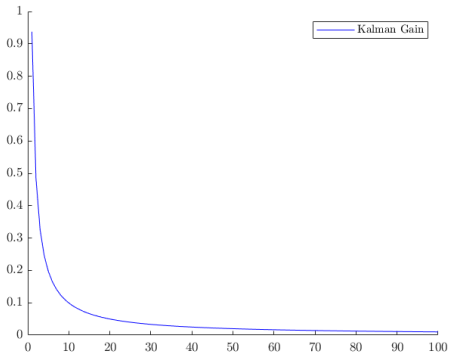

Figure 6. Kalman gain.

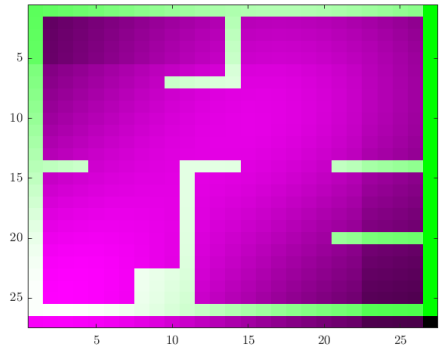

Figure 7. True underlying CO_2 distribution in the map.

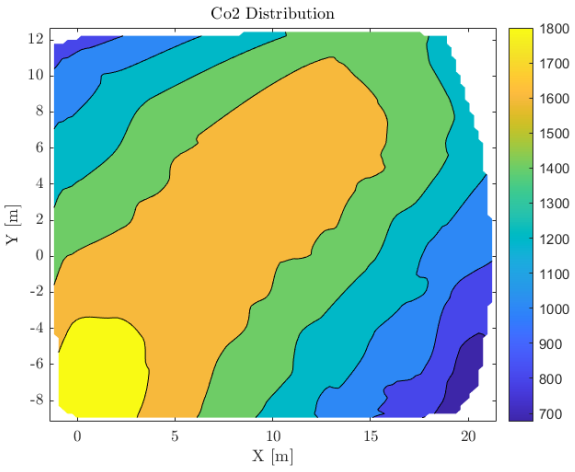

Figure 8. Reconstructed CO_2 distribution by using Kalman filter on the collected data.

actual map layout itself given in terms of world absolute coordinates. The other is the absolute position of the robot in the map. So, when the function $LiDARScan$ is invoked, the simulator can check what is around the absolute location of the rover. This way, the algorithm returns a tuple containing the distances and the angles of the surroundings in the cone of view of the robot.

Therefore, each robot holds a property which is inherent of the absolute position of the robot in the map, called $obj.absoluteLoc$. This property is always updated by an amount equal to the relative movement that the robot has made in the shared MDS.

This is the only reason why the respective absolute position is kept in memory for each robot: otherwise it would not be possible to perform scans with matlab's simulated LiDAR. This problem, of course, would not occur in the real world situations.

5. Exploration

Once a robot has performed a scan it moves to a new position for a new acquisition. To choose where to go, it selects a random free point within a certain radius from it and it use the A-star method to find the path to follow to reach the desired position. The A-star (A*) method is a path finding algorithm widely used in artificial intelligence and robotics to find the shortest path between two points in a graph or in a map of an environment. The algorithm combines two search approaches: breadth-based search and heuristic-based search. In other words, the method explores the nodes systematically to find the shortest path, but also uses a heuristic to evaluate the goodness of the nodes in order to choose the most efficient path.
The evaluation function so takes into account both the actual cost of the path so far (i.e. cost $g(n)$) and the estimated heuristic distance from the current node to the destination node (i.e. cost $h(n)$). This evaluation function is expressed as:

$$f(n) = g(n) + h(n)$$

A-star proceeds iteratively, exploring the nodes of the graph in increasing order of $f(n)$ until it reaches the destination node. While browsing, the algorithm maintains a list of open nodes and a list of closed nodes, where open nodes are the nodes that have yet to be explored and closed nodes are the nodes that have already been explored.

This method is very efficient in finding the shortest path in environments with many possible roads and obstacles so it's perfect for the conditions of this project, where the environment is unknown.

6. Results

When all the robots have finished exploring, they share their acquisitions to create an overall map of the areas explored on which the distribution of CO_2 is applied. The obtained results are satisfying. In Fig. 9 is possible to see

how well the robots explore the map and how the data are put together.

It is important to highlight how the coordinate system is different from that of the map in the real world (Fig. 1). This is due to the fact that all the acquisitions have been associated with the relative position of the robots within the reference system created via MDS and that therefore the origin corresponds to the position of the first robot instead of the bottom left corner.

The robots can reconstruct the CO_2 map well. By interpolating the data, the reconstruction is very similar to the original data. In Fig. 8 the yellow and orange area corresponds to the diagonal in Fig. 7 which it can be seen as the source of the gas. Moreover, Fig. 10 represents how after 30 steps of exploration the reconstruction of the CO_2 distribution is overlapped with the discovered map, and is very close to the true distribution shown in Fig. 7.

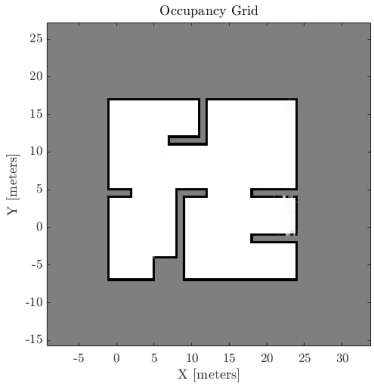

Figure 9. Reconstructed explored map.

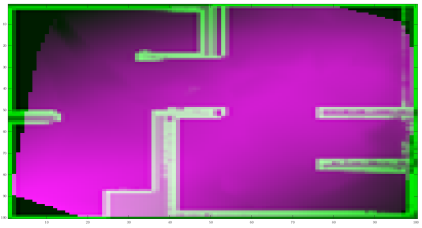

Figure 10. Reconstructed C02 distribution over the explored map.

7. Future Updates

The project can be expanded and improved from various points of view.

The first and most obvious is to replace rovers with drones so as to allow more efficient exploration of larger and three-dimensional areas.

Another idea is to replace the MDS method for creating the reference system with a partial MDS. This method allows the introduction of initially unknown nodes into an existing map without having to repeat the entire analysis from scratch. Through this technique it would therefore be possible to expand the exploration area and ensure that even if the robots are initially unable to enter the reference system due to the distance and the impossibility of communicating, they can do so later once they have entered the communication range with the other robots.

8. Conclusion

In this project 4 rovers have been placed in an unknown arena. Thanks to the implementation of the MDS, these robots are able to create a reference system and then freely explore the map.

During the exploration at the same time they also analyze the air quality through a CO_2 sensor filtering the measurements through a Kalman filter so as to correct them. This work therefore focuses particularly precisely on the implementation of the MDS algorithm and the Kalman filter being key points.

From the results obtained, it can be seen that the preestablished objectives have been met, however, leaving the doors open to future updates and improvements.

References

[1] Alex Becker. Kalman filter tutorial. https://www.kalmanfilter.net/default.aspx. 4, 5

[2] Qinyin Chen, Y. Hu, and Zhe Chen. Node localization algorithm of wireless sensor networks for large electrical equipment monitoring application. In *2016 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC)*, pages 390–397, 2016. 1

[3] Zhang-Xin Chen, He-Wen Wei, Qun Wan, Shang-Fu Ye, and Wan-Lin Yang. A supplement to multidimensional scaling framework for mobile location: A unified view. *IEEE Transactions on Signal Processing*, 57(5):2030–2034, 2009. 3

[4] Yingqiang Ding, Hua Tian, and Gangtao Han. A distributed node localization algorithm for wireless sensor network based on mds and sdp. In *2012 International Conference on Computer Science and Electronics Engineering*, volume 1, pages 624–628, 2012. 1

[5] Dan Jia, Weihua Li, Peng Wang, Xinxi Feng, Hongyan Li, and Zhiqiang Jiao. An advanced distributed mds-map localization algorithm with improved merging strategy. In *2016 IEEE International Conference on Information and Automation (ICIA)*, pages 1980–1985, 2016. 1

[6] Shane Mueller. Distance, similarity and multidimensional scaling. https://pages.mtu.edu/ shanem/psy5220/daily/Day16/MDS.html. 3