FINAL REPORT

Tutor: GOUBAULT Eric (LIX)
Coordinator: SCHAEFFER Gilles (LIX)
PSC: Swarm Rescue

12 mars 2024

Otávio Ribas, Coralie Tonle, El-Mehdi Nezahi,
Gabriel Pereira de Carvalho, Grégoire Michel-Delétie

EXECUTIVE SUMMARY

Our project is centered on the Swarm-Rescue competition organised by the Centre Interdisciplinaire de Défense et de Sécurité (CIEDS) at the Paris Institute of Technology and the Agence d'Innovation de Défense (AID).

The goal of the PSC is to design a control algorithm for a swarm of drones performing a rescue mission in an unknown environment. The competition provides a simulation environment where all the code is executed and where the solutions proposed by the participating teams are evaluated.

As described in the competition's website, this problem has real world applications :

The goal of the mission is to explore an unknown, difficult to access, potentially dangerous area, to search for people and guide them out of the area. The typical use case consists of investigating the basement of a collapsed building in the dark or smoke, in order to locate trapped people and rescue them by guiding them to the exit.

This report has a couple of different objectives :
— present intermediate results and the current state of the group's work
— describe the organisation of the group's work and weekly meetings
— describe difficulties and challenges currently being tackled

Github repository and Youtube video simulations

The project's code can be accessed on Github and videos of the simulations performed can be accessed on Youtube.

Table des matières

Chapitre 1
Introduction : Swarm Rescue Competition

The Swarm Rescue competition is organised in three stages. The goal is to provide a road map for the teams to incrementally build a solution the challenge.

— **5 December 2023** : First evaluation by a competition jury
— **30 January 2024** : Second evaluation by a competition jury
— **14 March 2024** : Final of the competition

Let's recall in more detail each of the three assessments that make up the competition.

First Evaluation : Evaluation of Navigation Skills

— **Scenario** : The environment used is relatively simple, with few obstacles, a single injured person, a single drone and an area of GPS loss.
— **Objective** : To measure navigation capabilities.
— **Evaluation Criteria** : Teams will be evaluated on the time taken to get the injured person to the rescue zone, both with and without the GPS loss zone.

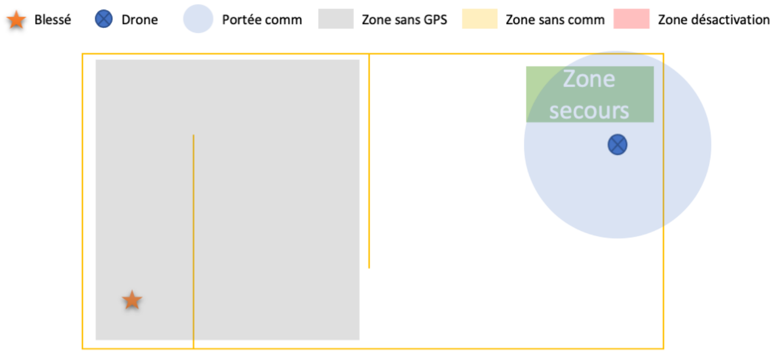


Figure 1.1 – Example scenario for the first scale

Second Assessment : Assessment of Coordination and Communication Skills

- **Scenario** : This scenario involves a simple environment to assess coordination and communication capabilities with 10 drones, many injured people, a large unobstructed map, and areas of communication loss and drone loss.
- **Objective** : To assess coordination and communication capabilities.
- **Evaluation Criteria** : Teams will be evaluated on the time taken to get all injured people to the rescue area, both with and without the communication and drone loss zones.

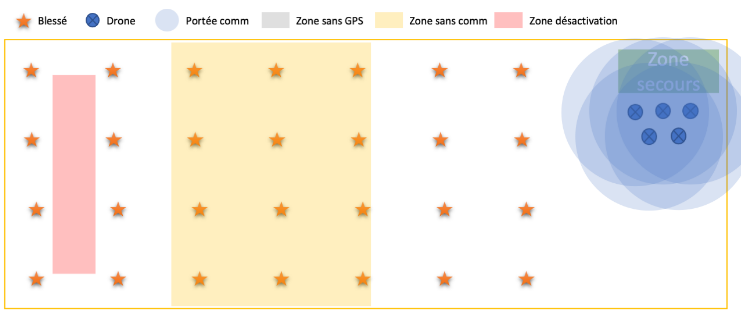


Figure 1.2 – Example scenario for the second scale

Final evaluation : Final of the competition

- **Scenario** : The final evaluation will take place on maps unknown to the participants, containing all the difficulties of the complete scenario.
- **Objective** : To provide an overall assessment taking into account the percentage of people rescued, the percentage of the map explored and the percentage of time remaining in relation to the allotted time once all the people have been rescued.
- **Evaluation Criteria** : The evaluation will encompass a variety of qualitative and quantitative criteria including :
  1. Quality of presentation
  2. Quality of answers to questions
  3. Operational credibility of the solution
  4. Respect for the spirit of the challenge

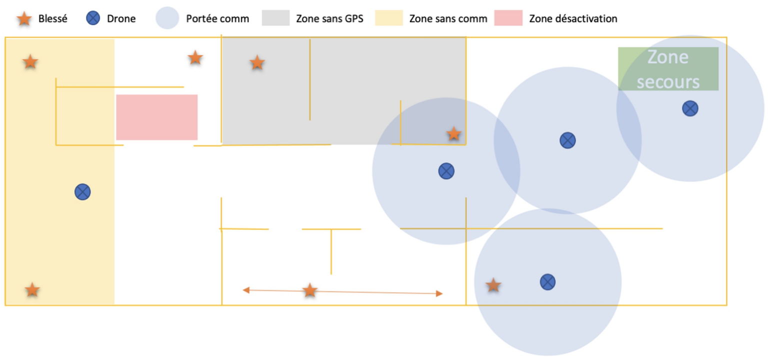


Figure 1.3 – Sample scenario for the competition final

Chapitre 2
Finite State Machine

In this chapter we begin describing the logic of our control algorithm. Each drone executes an instance of this algorithm, meaning each drone must be able to work independently but can use communication to try to improve its performance.

The first step in developing the control algorithm was dividing the problem into subproblems. Then, we must decide the conditions to transition between these subproblems, so the drone can know at each loop of the algorithm which problem it is supposed to be solving. We propose a finite state machine(FSM) where each state corresponds to a subproblem the drone must solve over the course of the simulation.

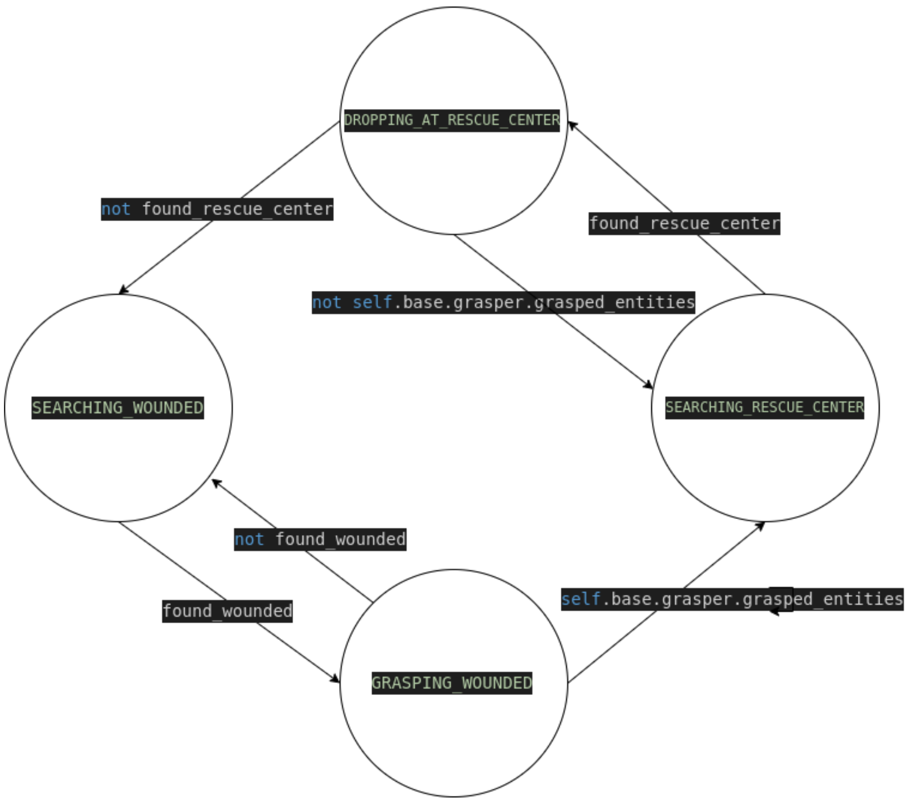


Figure 2.1 – Finite State Machine logic implemented for each drone

The FSM in figure 2.1 is a loop composed of four states :

1. `SEARCHING_WOUNDED` : the initial state. Each drone must explore the map until it finds a wounded person.
2. `GRASPING_WOUNDED` : once the semantic sensor has detected a wounded person, the drone must move towards this person and grasp him.

   It is possible that two drones detect and move towards a same person. Communication should be used in this case and only one drone should grasp the wounded person. For this reason, the FSM includes a transition from `GRASPING_WOUNDED` back to `SEARCHING_WOUNDED`.

3. `SEARCHING_RESCUE_CENTER` : once the has grasped a wounded person, it must move back to the rescue center.
4. `DROPPING_AT_RESCUE_CENTER` : once the semantic sensor has detected the rescue center, the drone must move towards it. After the person's dropped at the rescue center we deactivate the flag `found_rescue_center` to transition back to
5. `SEARCHING_WOUNDED`. The loop begins again until the end of the simulation.

Chapitre 3

Moving towards landmark

In the finite state machine, the semantic sensor plays a very important part. Each drone uses its semantic sensor to identify landmarks in the environment, which trigger transitions in the state machine. In the state `SEARCHING_WOUNDED`, we are interested in detecting a wounded person. In the state `SEARCHING_RESCUE_CENTER`, we are interested in detecting the rescue center so we can drop the wounded person currently grasped. In either state, we are dealing with the same task : moving towards a near landmark detected by the semantic sensor. In this section, we present a solution to this problem based on the **P-Only controller**, a simple algorithm in the PID family.

In the code, we have defined a function `process_semantic_sensor` responsible for updating the flags `found_wounded` and `found_rescue_center` as well as interpreting the semantic sensor data in order to recommend a command for the drone.

3.1 THE P-ONLY CONTROLLER

Our goal is to use the P-only controller to calculate the optimal angle for the drone to reach the landmark. We define the drone's speed in function of the estimated optimal angle. If the drone needs a big rotation, we use a small speed of 0.2 to prevent it from drifting too far from the landmark, however if the drone needs a sufficiently small rotation we can move towards the landmark at a higher speed of 0.5. These values were found to work well empirically.

Let the drone's compass angle at time t is $\theta(t)$ and let the angle of the line joining the angle and the landmark be $r(t)$. Our goal is to make $\theta(t) = r(t)$ so the drone can move towards the landmark. At each step, the control algorithm can measure the error $e(t) = r(t) - \theta(t)$ and we can use this information to reduce the error at each iteration. This design is know in control theory as a **feedback loop**.

The proportional gain K_p of the P-controller is a tuning parameter that determines how much the controller should respond to the current error. In our case this constant was set empirically based on our simulation results. The control output $u(t)$ of the P-only controller is then

$$u(t) = K_p \cdot e(t).$$

Chapitre 4
Approach 1 : Hilbert Curves

A space-filling curve is a one-dimensional curve which passes through every point of an n-dimensional region. Hilbert's space-filling curve covers all points with integer coordiantes in a 2-dimensional square grid of size $2^m \times 2^m (m \in \mathbb{N})$. In order to plan a path for the robot that is guaranteed to cover the grid, we can use Hilbert curves. This approach can be refined further into complex exploration strategies [1] [2].

The Hilbert curve is constructed using a recursive process that iteratively builds smaller curves and then connects them in a specific order. The construction involves dividing the space into four quadrants and recursively building smaller Hilbert curves within each quadrant.

4.1 BUILDING A HILBERT CURVE

1. **Base Case :** The base case is a straight line segment connecting two endpoints.
2. **Iteration :** At each iteration, the curve is divided into four quadrants, and a smaller Hilbert curve is constructed recursively within each quadrant.
3. **Rotation and Connection :** The smaller Hilbert curves are rotated and connected to form a larger curve. The order of connecting quadrants determines the overall shape.

Space-Filling Properties :

The space-filling properties of the Hilbert curve are crucial for its applications.

1. **Every Point is Visited :** The Hilbert curve passes through every point in the space without leaving any gaps.
2. **Proximity in 1D Implies Proximity in 2D :** Points that are close to each other along the curve in the one-dimensional space are also close to each other in the two-dimensional space.

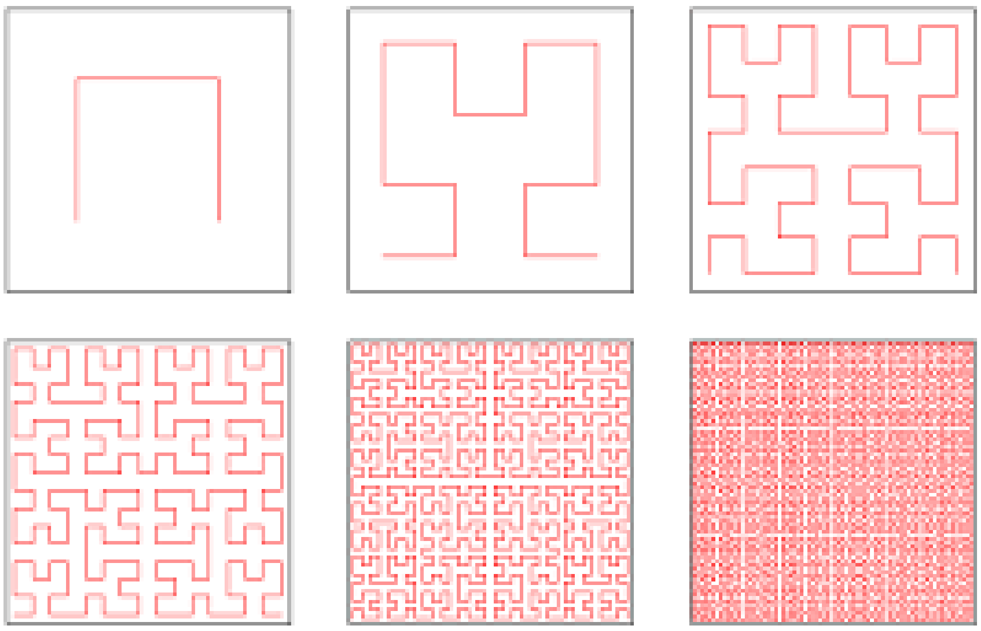


Figure 4.1 – Hilbert curves for $n = 1$ up to 6

Mathematical Representation :

The mathematical representation of the Hilbert curve involves recursive expressions and transformations.

Recursive Function :

```
function hilbert_curve(order, angle, length):
    if order is 0:
        draw a straight line of length
    else:
        turn left by angle
        hilbert_curve(order - 1, -angle, length)
        move forward by length
        turn right by angle
        hilbert_curve(order - 1, angle, length)
        move forward by length
        hilbert_curve(order - 1, angle, length)
        turn right by angle
        move forward by length
        hilbert_curve(order - 1, -angle, length)
        turn left by angle
```

4.2 EXPLORATION ALGORITHM

In order to find the wounded person, our strategy is to draw a Hilbert curve over the map and follow it until the semantic sensor detects the wounded. At each step of our algorithm, we compute points on the Hilbert curve adjacent to the current position. We select an unvisited point from this sample and then move in its direction.

In collision-free space, we can follow simply the Hilbert curve, however we also need a collision avoidance strategy. It is possible that at a certain step of the algorithm, there are no points on the Hilbert curve we can visit in collision-free space. When this happens, our strategy is to keep a historic of GPS positions and move away from these recent positions in hope of finding a new segment of the curve we can continue to explore.

4.3 TESTING

This approach has been tested on the Swarm Rescue competition's simulation environment. Video of the simulation can be downloaded from the project's Github repository.

Chapitre 5

Approach 2 : Rapidly Exploring Random Trees

5.1 BUILDING THE RRT (EXPLORATION STAGE)

The idea for the RRT exploration algorithm is similar to a DFS (depth-first search). At each step of the algorithm, given the drone's position we calculate neighboring points in collision-free space. The collision-free space is defined using the distance measurements from the lidar, directions with a sufficiently big distance measurement are considered valid. We then generate points in these directions to build the set of points denoted as collision-free space.

Then, the RRT samples a random point in collision-free space (the sampling process is called **steering** [3]) and we command the drone to move towards this point. Each time an unexplored point is discovered, we connect it the previous point in the drone's trajectory. The RRT gives us two interesting properties [3] :

— the expansion of the RRT is biased toward unexplored portions of the map ;

— the RRT always remains connected, avoiding obstacles in the map.

Once the semantic sensor detects a person, we no longer need to sample random points in order to define the trajectory. Instead we follow the commands given by the P-only controller based on the semantic sensor data. However, we continue keeping track of the drone's position and adding these points to the tree.

In the code, we created a RRT class that abstracts all the tree specific functions which are later used in the drone's control algorithm. The functions implemented are :

— creating new nodes

— adding edges between two nodes

— checking if a given node belongs to the tree (which means it's already been explored)

— retrieving a node object given its point's coordinates

— sampling a point in collision-free space (**steering**)

— computing the lowest common ancestor (LCA) of two tree nodes

— building a path between two tree nodes

The lowest common ancestor (LCA) function is used to construct paths between tree nodes

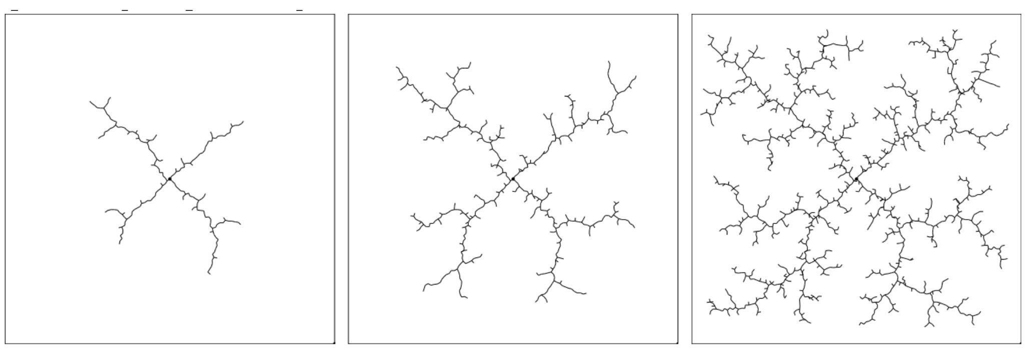


Figure 5.1 – Evolution of an RRT over time for a 100×100 grid with $x_{init} = (50, 50)$

u, v. Let $w = \text{LCA}(u, v)$, we have $\text{PATH}(u, v) = \text{PATH}(u, w) \oplus \text{PATH}(w, v)$. We note that efficient $O(\log |V|)$ algorithms exist for LCA queries such as Tarjan's off-line algorithm and the binary-lifting algorithm. But since we are interested in explicitly reconstructing the list of tree nodes on the path, we adopted the simples $O(|V|)$ approach.

5.2 Using RRT to construct paths : Bézier curves

Once a drone has found a person, the goal is to reach the rescue center as fast as possible. If a node on the RRT represents the rescue center, we can then follow the unique path on the tree to reach the rescue center. However since the path discovered in the exploration stage is random, it is not very efficient. We need a strategy to build a smooth path from the set of points on the tree path.

We propose using Bézier curves[4], these curves are widely used in computer graphics to define a smooth, continuous curve given a set of discrete control points. The Bézier curve of degree n with control points $P_0, P_1, \ldots, P_n$ is defined by the formula :

$$B(t) = \sum_{i=0}^{n} P_i \cdot B_i^n(t)$$

where $B(t)$ is the Bézier curve function at parameter t, P_i are the control points, and $B_i^n(t)$ are the Bernstein basis polynomials.

The Bernstein basis polynomials are given by :

$$B_i^n(t) = \binom{n}{i} t^i (1-t)^{n-i}$$

where $\binom{n}{i}$ is the binomial coefficient, calculated as $\frac{n!}{i! \cdot (n-i)!}$, and t is the parameter ranging from 0 to 1.

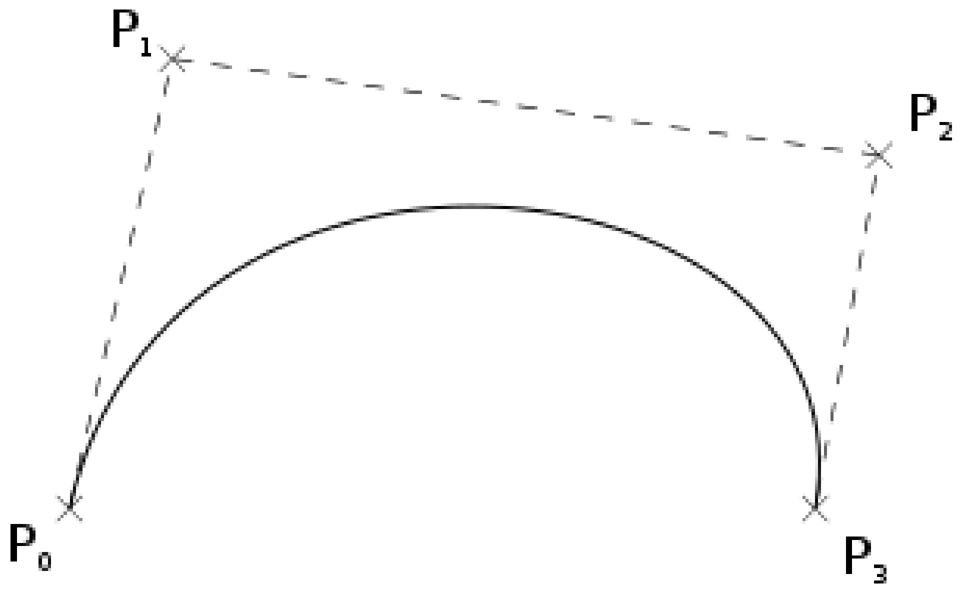


Figure 5.2 – Example of a Bézier curve with four control points

In order to explicitly build the path back to the Rescue Center, we need to compute an array of points on the Bézier curve given it's control points. To do this, we use Casteljau's algorithm which given control points $P_0, P_1, \ldots, P_n$ and the value of the curve's parameter t computes the point $B(t)$.

Step 1 : Initialization

Consider control points $P_0, P_1, \ldots, P_n$. Initialize $Q_i^0 = P_i$.

Step 2 : Recursive Interpolation

For each level k (where $k = 1, 2, \ldots, n$), calculate new points Q_i^k :

$$Q_i^k = (1-t) \cdot Q_i^{k-1} + t \cdot Q_{i+1}^{k-1}$$

This process is repeated until only one point remains, which is the final point on the Bézier curve.

Step 3 : Final Result

The point on the Bézier curve $B(t) = Q_0^n$.

Similar to our implementation strategy for the RRT, we created a Bézier curves class that abstracts the mathematics used to building the curve which is later used in the drone's control algorithm. The problem we aim to solve in this class in simple : build the Bézier curve given a set of n points (the list of tree nodes' coordinates built using the RRT).

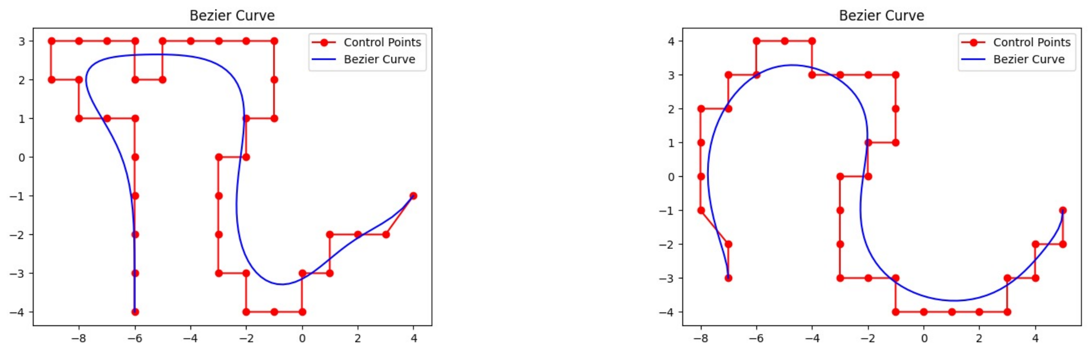


Figure 5.3 – Paths back to the Rescue Center built using Bézier curves

5.3 TESTING

The solution has been tested on several different maps. As discussed previously, video simulations can be found on Youtube The figure 5.5 illustrates the simulation results.

5.4 COMPARISON : HILBERT CURVE VS. RRT

In the previous chapter, we examined a simpler approach to solve the swarm rescue problem : following a Hilbert curve drawn over the given map. Using Hilbert curves, we made assumptions about the map size in order to build the curve in the initialization step. An advantage of the RRT approach is making no such assumption. We build a map online that gives us more information on the initially unknown region over time. Using the RRT's map, we were able to develop a strategy to improve performance on the way back to the rescue center.

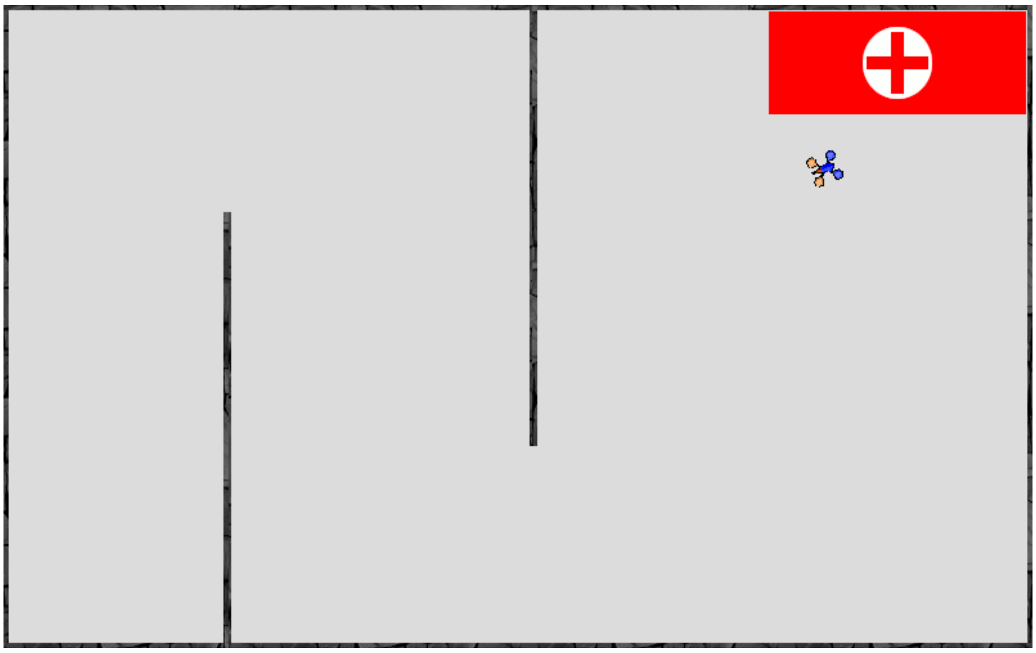


Figure 5.4 – Example of a map used for testing

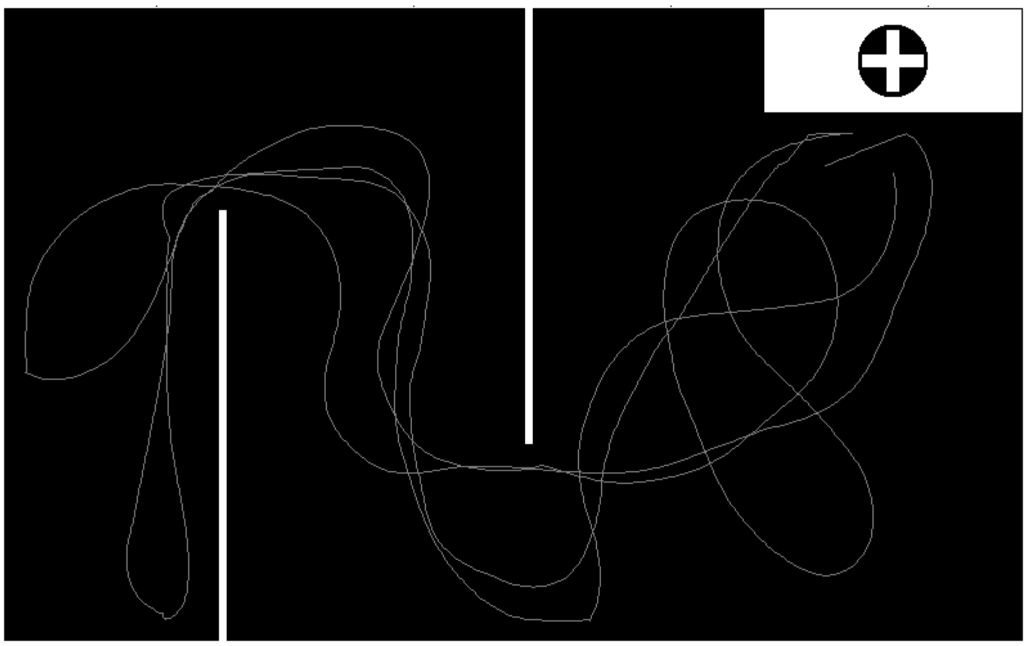


Figure 5.5 – Example of path followed by drone

Chapitre 6
Predicting position without GPS

Let's consider the RRT solution we have proposed. When the drone is looking for a wounded person, it is simultaneously tackling three different problems.

1. Mapping Problem : Building a map of the unknown region.
2. Localization Problem : Determining its position on this map.
3. Planning Problem : Choosing a strategy to explore the region.

For the moment, we are reliant on the GPS in order to localize the robot. In order to solve maps with no-GPS zones, we need to solve the localization problem without using data from the GPS. The mapping and planning procedures in our solution can stay the same.

6.1 APPROACH 1 : INTEGRATING ODOMETRY DATA

A first approach to replace the GPS data is using odometry to track the movement of the drone relative to its last position. Ideally, if we sum all the relative displacements given by the odometer, we should get a good estimation of the drone's position without using the GPS. However, the odometry data in the simulation environment has an unknown gaussian noise. This means that when integrating over this data, the error in each measurement adds up and can become quite big over time. This problem can be observed in figure 6.1.

In fact, the data from all the given sensors (lidar, semantic sensor, odometry) has an unknown gaussian noise. Thus, we cannot rely on a prediction given by a model based on any individual sensor, for a good accuracy we must combine these different predictions. This problem is called the **sensor fusion** problem. At this moment, we have not implemented a solution to this problem in our project yet.

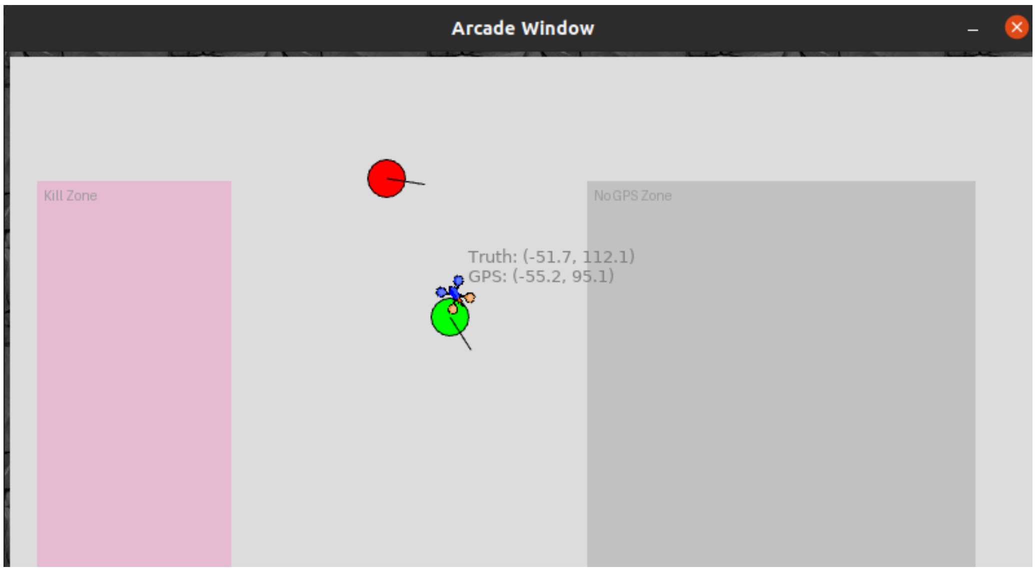


FIGURE 6.1 – The green circle represents GPS data, the red one represents the Odometry prediction.

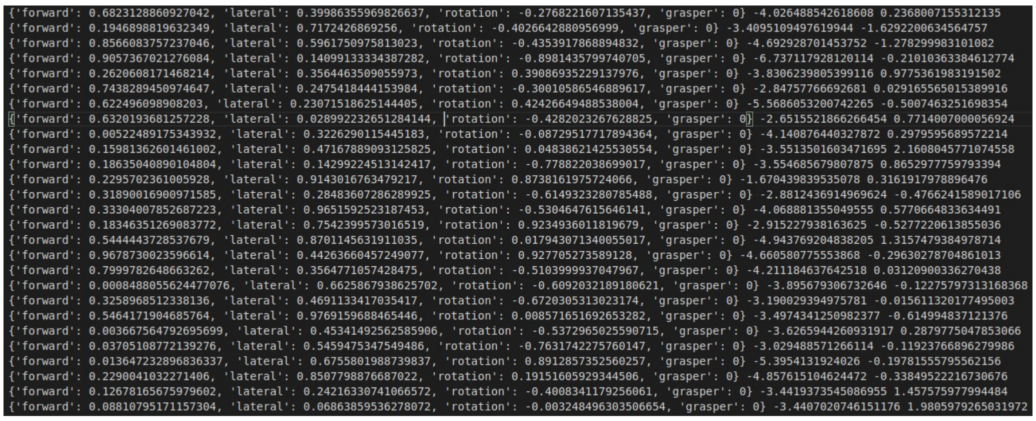
6.2 APPROACH 2 : USING A NEURAL NETWORK TO PREDICT POSITION

Using maps that do not contain no-GPS zones, we collected training data (X, Y) using the GPS where X represent the command issued by the control loop and Y represents the displacement $(\Delta x, \Delta y)$ measured in the next iteration. Then, we can use this data to train a **sequential neural network model** that can predict the drone's position in the no-GPS zone. We used purely random movements to collect the data shown in 6.2.

In order to minimize the overhead of the calculations used for prediction, we used **TensorFlow Lite** to export the trained model [5]. We also used post-training quantization to reduce the model's size while also improving CPU and hardware accelerator latency, with little degradation in model accuracy.

6.2.1 TRAINING THE MODEL

We trained the model on **Google Colab** using the Keras library from TensorFlow. Our dataset contained ≈ 4200 data points, we split the data into three parts :
— 60% training data ;
— 20% validation data and
— 20% testing data.



```
{'forward': 0.6823128860927042, 'lateral': 0.39986355969826637, 'rotation': -0.2768221607135437, 'grasper': 0} -4.026488542618608 0.2368007155312135
{'forward': 0.1946898819632349, 'lateral': 0.7172426869256, 'rotation': -0.4026642880956999, 'grasper': 0} -3.4095109497619944 -1.6292200634564757
{'forward': 0.8566083757237046, 'lateral': 0.5961750975813023, 'rotation': -0.4353917868894832, 'grasper': 0} -4.692928701453752 -1.278299983101082
{'forward': 0.9057367021276084, 'lateral': 0.14099133334387282, 'rotation': -0.8981435799740705, 'grasper': 0} -6.737117928120114 -0.21010363384612774
{'forward': 0.2620608171468214, 'lateral': 0.3564463509055973, 'rotation': 0.39086935229137976, 'grasper': 0} -3.8306239805399116 0.9775361983191502
{'forward': 0.7438289450974647, 'lateral': 0.2475418444153984, 'rotation': -0.30010586546889617, 'grasper': 0} -2.84757766692681 0.029165565015389916
{'forward': 0.622496098908203, 'lateral': 0.23071518625144405, 'rotation': 0.42426649488538004, 'grasper': 0} -5.5686053200742265 -0.5007463251698354
{'forward': 0.6320193681257228, 'lateral': 0.028992232651284144, 'rotation': -0.4282023267628825, 'grasper': 0} -2.6515521866266454 0.7714007000056924
{'forward': 0.00522489175343932, 'lateral': 0.3226290115445183, 'rotation': -0.08729517717894364, 'grasper': 0} -4.140876440327872 0.2979595689572214
{'forward': 0.15981362601461002, 'lateral': 0.47167889093125825, 'rotation': 0.04838621425530554, 'grasper': 0} -3.5513501603471695 2.1608045771074558
{'forward': 0.18635040890104804, 'lateral': 0.14299224513142417, 'rotation': -0.778822038699017, 'grasper': 0} -3.554685679807875 0.8652977759793394
{'forward': 0.2295702361005928, 'lateral': 0.9143016763479217, 'rotation': 0.8738161975724066, 'grasper': 0} -1.670439839535078 0.3161917978896476
{'forward': 0.31890016900971585, 'lateral': 0.28483607286289925, 'rotation': -0.6149323280785488, 'grasper': 0} -2.8812436914969624 -0.4766241589017106
{'forward': 0.33304007852687223, 'lateral': 0.9651592523187453, 'rotation': -0.5304647615646141, 'grasper': 0} -4.068881355049555 0.5770664833634491
{'forward': 0.18346351269083772, 'lateral': 0.7542399573016519, 'rotation': 0.9234936011819679, 'grasper': 0} -2.915227938163625 -0.5277220613855036
{'forward': 0.5444443728537679, 'lateral': 0.8701145631911035, 'rotation': 0.017943071340055017, 'grasper': 0} -4.943769204838205 1.3157479384978714
{'forward': 0.9678730023596614, 'lateral': 0.44263660457249077, 'rotation': 0.927705273589128, 'grasper': 0} -4.660580775553868 -0.29630278704861013
{'forward': 0.7999782648663262, 'lateral': 0.3564771057428475, 'rotation': -0.5103999937047967, 'grasper': 0} -4.211184637642518 0.03120900336270438
{'forward': 0.0008488055624477076, 'lateral': 0.6625867938625702, 'rotation': -0.6092032189180621, 'grasper': 0} -3.895679306732646 -0.12275797313168368
{'forward': 0.3258968512338136, 'lateral': 0.4691133417035417, 'rotation': -0.6720305313023174, 'grasper': 0} -3.190029394975781 -0.015611320177495003
{'forward': 0.5464171904685764, 'lateral': 0.9769159688465446, 'rotation': 0.008571651692653282, 'grasper': 0} -3.4974341250982377 -0.614994837121376
{'forward': 0.003667564792695699, 'lateral': 0.45341492562585906, 'rotation': -0.5372965025590715, 'grasper': 0} -3.6265944260931917 0.2879775047853066
{'forward': 0.03705108772139276, 'lateral': 0.5459475347549486, 'rotation': -0.7631742275760147, 'grasper': 0} -3.029488571266114 -0.11923766896279986
{'forward': 0.013647232896836337, 'lateral': 0.6755801988739837, 'rotation': 0.8912857352560257, 'grasper': 0} -5.3954131924026 -0.19781555795562156
{'forward': 0.2290041032271406, 'lateral': 0.8507798876687022, 'rotation': 0.19151605929344506, 'grasper': 0} -4.857615104624472 -0.3384952216730676
{'forward': 0.12678165675979602, 'lateral': 0.24216330741066572, 'rotation': -0.4008341179256061, 'grasper': 0} -3.4419373545086955 1.457575977994484
{'forward': 0.08810795171157304, 'lateral': 0.06863859536278072, 'rotation': -0.003248496303506654, 'grasper': 0} -3.4407020746151176 1.9805979265031972
```

Figure 6.2 – Training data for the regression model

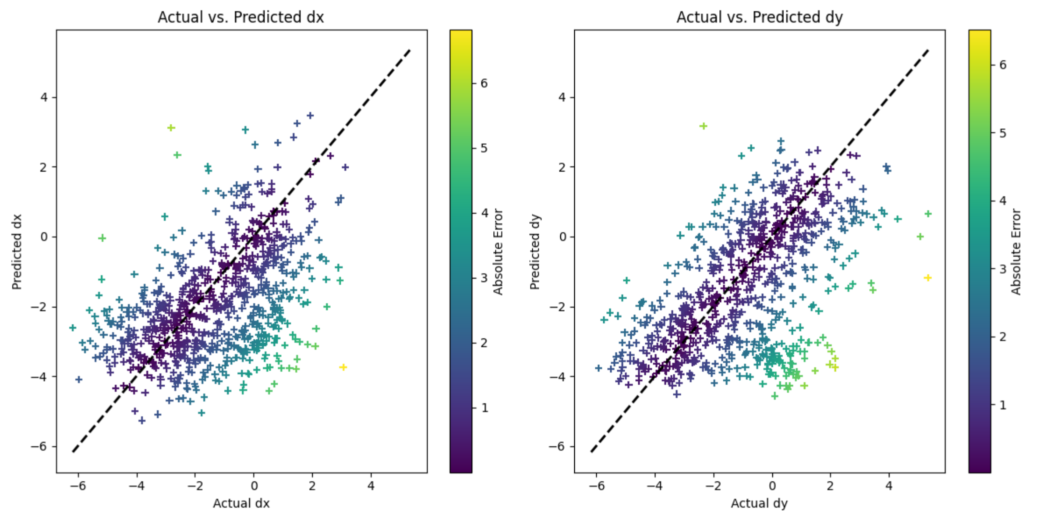


Figure 6.3 – Results on testing data

Before hand, we performed a random shuffle to ensure that each part contained the same distribution of information. Using this methodology, we can check training and validation loss over time to be sure that the model is not overfitting. In addition, the testing data allows us to verify if the model can generalize well into unknown data.

6.2.2 Testing results

In figure 6.3, we can see the distribution of $(\Delta x_{real}, \Delta x_{predicted})$ and $(\Delta y_{real}, \Delta y_{predicted})$ on the testing data. We obtained the following metrics from this testing procedure :

— loss = 3.5974

— mae = 1.4789 (mean absolute error)

6.2.3 Exporting with TensorFlow Lite

TensorFlow Lite converts models into a special, space-efficient format. The optimizations applied by the converter reduce model size and make it run faster. Since the computational overhead can be significant when running the simulation with multiple different drones, we decided this was a good approach to optimize the model's performance. Our final version of the model is only **11 kilobytes** in size.

6.3 TESTING

This solution for no-GPS zones was integrated into the project and has been tested on several different maps. As discussed previously, video simulations can be found on Youtube.

Chapitre 7

Generating random maps

In order to train a learning algorithm, we needed to have a lot of data in the form of maps to train the algorithm on. We therefore decided to make a random generator of maps. We decided to make a modular generator using templates applicable on any chosen area in order to mix different styles of buildings. All areas are surrounded by walls by default. For now the only template available is a hallway. The corresponding function can generate a hallway with a chosen size, position, as well as orientation.

For example :

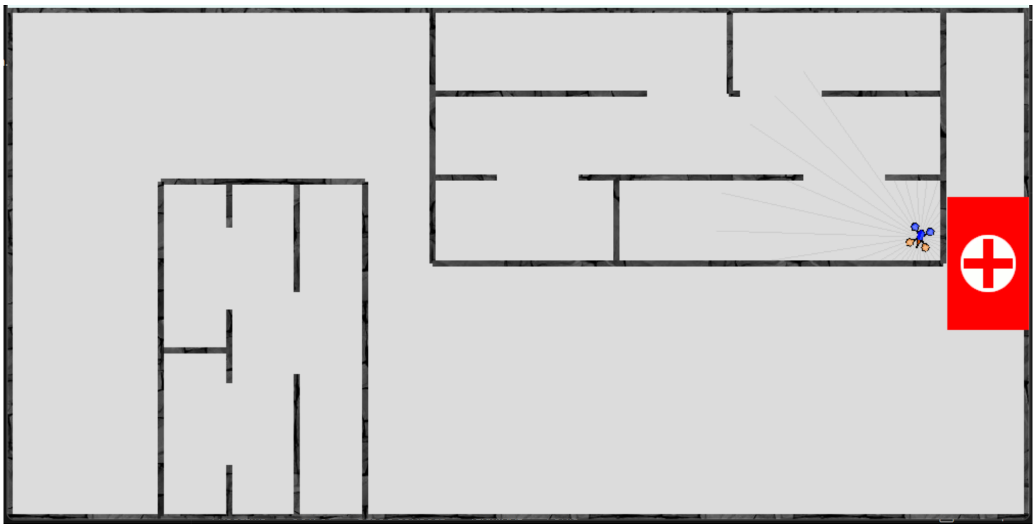


Figure 7.1 – An example with two hallways

In order to keep the map connected, designated areas can be opened on demand. Here is the result with three hallways : two horizontal on the left and a vertical one on the right. At the end of each hallways was inserted a passage for the drone. A given amount people to rescue for each area was also placed randomly (here 20 on the top left, 6 on the bottom left and 10 on the right).

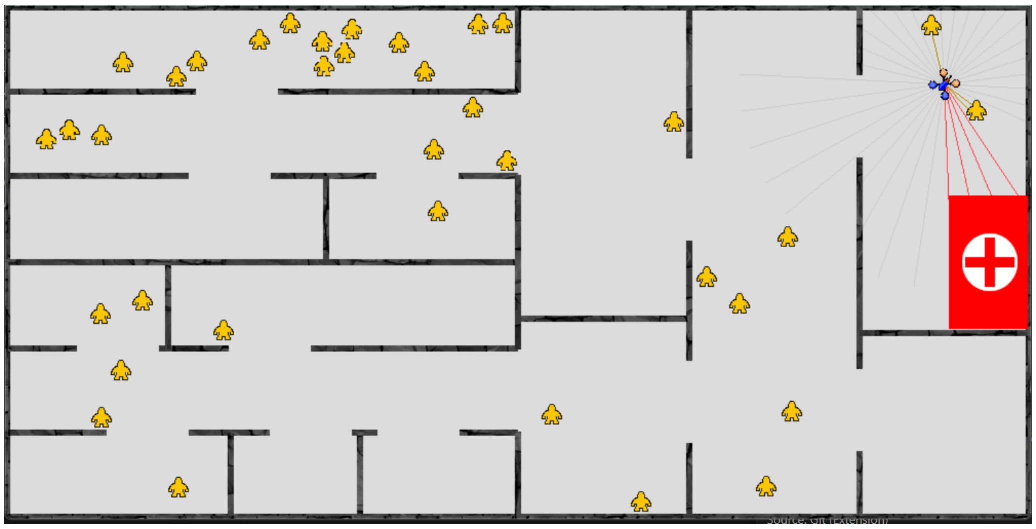


Figure 7.2 – Example of map

Chapitre 8

Organization of group work

8.1 MEETING LOCATION

In order to work on a regular basis, we decided to meet up every Wednesday during our PSC time in a co-working room in the campus of Ecole polytechnique. We are mainly working in the DrahiX where we reserved the co-working room which can be seen at the left of figure 1.1 or at the Ecole polytechnique's library. However, the biggest problem we faced at the library is the crowding of students Wednesday afternoon since there is not a reservation system there.

FIGURE 8.1 – Weekly group meetings at polytechnique

Bibliographie

[1] Maulik C. BHATT Anant A. JOSHI and Arpita SINHA. Modification of hilbert's space-filling curve to avoid obstacles : A robotic path-planning strategy. *arXiv Robotics*, 2019.

[2] Arpita SINHA Ashay WAKODE. Online obstacle evasion with space-filling curves. *arXiv Robotics*, 2023.

[3] Steven M. Lavalle. *PLANNING ALGORITHMS*. Cambridge University Press, 2006.

[4] DUNCAN Marsh. *Applied geometry for computer graphics and CAD. — 2nd ed. — (Springer undergraduate mathematics series)*. Springer, 2005.

[5] SITUNAYAKE Daniel WARDEN Pete. *TinyML : Machine Learning with TensorFlow Lite on Arduino and Ultra-Low-Power Microcontrollers*. O'Reilly, 2020.