UNIVERSITY OF BOLOGNA
School Of Engineering

Multi-Agent Systems - Course Project

Master's Degree in Artificial Intelligence

Distributed Multi-Robot Search and Rescue Operations Using Aggregative Optimization in a Multi-Room House Environment

Professors:
Andrea Omicini
Roberta Calegari

Student:
Andrea Perna

Academic year 2023/2024

Abstract

This project presents a multi-agent system for distributed multi-robot search and rescue (SAR) operations in a multi-room house environment. The system leverages a distributed aggregative optimization algorithm executed by BDI (Beliefs, Desires, Intentions) agents. These agents maintain formation and reach consensus over the gradient of the cost function and the formation's barycenter, enabling efficient exploration and rescue missions. The robots navigate the house to locate survivors, escort them to the exit while avoiding obstacles, and subsequently resume exploration. An ambulance transports rescued individuals to a hospital, ensuring their safety. This implementation demonstrates the effectiveness of autonomous distributed multi-robot systems in dynamic and complex environments for critical SAR missions.

Contents

Introduction

The field of multi-agent systems (MAS) is crucial in robotics, enabling groups of autonomous agents to collaborate on complex tasks. In a multi-robot system, each robot acts as an individual agent, using collective intelligence to tackle tasks that would challenge a single robot, resulting in increased efficiency, redundancy, and flexibility. Agents in MAS operate autonomously, and one effective way to model this behavior is through the Belief-Desire-Intention (BDI) framework, by sticking with the strong agency's definition. This framework, which simulates human decision-making by maintaining beliefs, generating desires, and forming intentions to achieve goals, endows robots with intelligent, goal-oriented behavior. In distributed swarm robotics, optimization and consensus theory play a crucial role. Through local communication, robots reach consensus on parameters like formation barycenter and cost, enabling coordinated actions essential for tasks requiring synchronization and decentralized decision-making. MAS maintain a clear distinction between the environment and the agents, with the environment providing context and agents dynamically adapting. In urban search and rescue (USAR) operations, robotic swarms navigate complex environments using distributed algorithms for coordination and task allocation. The project employs distributed aggregative optimization and BDI robotic agents to simulate search and rescue (SAR) operations in a multi-room house, aiming to locate and escort survivors to safety. This report provides an overview of the system's development and demonstrates its effectiveness through detailed simulations in Python.

(a) Burning House

(b) Collapsed House

Figure 1: Project Motivations

Chapter 1

Distributed Aggregative Optimization Algorithm

1.1 Theoretical Framework

The distributed aggregative optimization algorithm is essential for optimizing global objectives while managing local, agent-specific cost functions independently. This fully distributed approach ensures scalability through local interactions and facilitates coordination by achieving consensus on shared variables. By balancing local and global objectives, it allows agents to minimize personal cost functions without compromising overall system efficiency. This makes it crucial for complex, real-world applications requiring flexibility and reliability, particularly in multi-agent systems (MAS) for efficient distributed operations.

1.1.1 Consensus Theory

Consensus theory is fundamental in MAS and distributed optimization, enabling agents to agree on certain quantities through local interactions and iterative information exchange. This consensus process ensures that agents' states converge to a common value despite having only local information. The general form of a discrete-time consensus algorithm is:

$$z_i^{k+1} = \sum_{j \in \mathcal{N}_i^{\text{out}}} a_{ij} z_j^k \tag{1.1}$$

where z_i^k and z_j^k represent the states of agents i and j at time k, a_{ij} are the non-negative communication weights between nodes in the digraph (directed graph) G, and $\mathcal{N}_i^{\text{out}}$ is the set of out-neighbors of i, i.e., those nodes i can directly communicate with. For a fixed communication topology, this system is modelled by a linear time-invariant (LTI) system:

$$z^{k+1} = Az^k \tag{1.2}$$

where z^k is the vector of all agents' states and A is the weighted adjacency matrix of the communication graph. The extended formulation of this dynamical system follows:

$$\begin{pmatrix} z_1^{k+1} \\ z_2^{k+1} \\ \vdots \\ z_N^{k+1} \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1N} \\ a_{21} & a_{22} & \cdots & a_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ a_{N1} & a_{N2} & \cdots & a_{NN} \end{pmatrix} \begin{pmatrix} z_1^k \\ z_2^k \\ \vdots \\ z_N^k \end{pmatrix} \tag{1.3}$$

Therefore, the adjacency matrix of a network of N agents is always square, i.e., $A \in \mathbb{R}^{N \times N}$.

A key property of the adjacency matrix A in graph networks is stochasticity. The A matrix is row-stochastic if each row sums to 1, i.e., $A\mathbf{1} = \mathbf{1}$, and column-stochastic if each column sums to 1, i.e., $\mathbf{1}^T A = \mathbf{1}^T$. A is is doubly stochastic if both conditions hold, hence:

$$\sum_{j=1}^{N} a_{ij} = 1 \quad \text{and} \quad \sum_{i=1}^{N} a_{ij} = 1 \quad \text{for all } i, j$$

Doubly stochastic matrices ensure balanced influence among agents, crucial for consensus. This property underlies the following theorem, on which the whole project is based.

Theorem (Consensus): Consider a DT averaging system with associated digraph G and row-stochastic weighted adjacency matrix A. Assume G to be strongly connected and aperiodic, then:

1. There exists a left eigenvector $w \in \mathbb{R}^N$ of A, $w > 0$ (i.e., with positive components $w_i > 0$ for all $i = 1, \ldots, N$) such that:

$$\lim_{k\to\infty} z^k = \mathbf{1}\frac{w^\top z^0}{w^\top 1} = \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix} \frac{\sum_{i=1}^{N} w_i z_i^0}{\sum_{i=1}^{N} w_i} \tag{1.4}$$

   i.e., consensus is reached to $\frac{\sum_{i=1}^{N} w_i z_i^0}{\sum_{i=1}^{N} w_i}$.

2. If additionally A is doubly stochastic, then

$$\lim_{k\to\infty} z^k = \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix} \frac{\sum_{i=1}^{N} z_i^0}{N} \tag{1.5}$$

   i.e., average consensus is reached.

The Consensus Theorem ensures agents converge to a common value if the adjacency matrix A is doubly stochastic and the associated graph G is strongly connected and aperiodic. Doubly stochasticity guarantees average consensus, where the final state is the average of initial states, ensuring balanced influence and stable convergence. This property enhances robustness and efficiency in distributed systems, crucial for tasks like formation control and sensor fusion. An important lemma supporting this theorem follows.

Lemma. Let A be a row-stochastic matrix and G the associated digraph. If G is strongly connected and aperiodic, then:

1. the eigenvalue $\lambda = 1$ is simple;
2. all the other eigenvalues μ satisfy $|\mu| < 1$.

This lemma implies that the matrix A has a simple largest eigenvalue equal to 1, ensuring a unique consensus value and stable convergence. The simplicity of the eigenvalue 1 implies that the influence of initial conditions decays over time, leading the system to converge to a steady state. Weights a_{ij} are crucial as they determine the convergence rate and robustness of the consensus algorithm. Properly chosen weights can accelerate convergence and improve the system's resilience to changes in network topology. They balance the influence each agent has on its neighbors and vice versa, facilitating efficient information propagation and agreement among the agents in the distributed network.

1.1.2 Optimization Theory

Optimization is a fundamental discipline in mathematics and engineering, aimed at selecting the best option from a set of available alternatives by minimizing or maximizing an objective function $\ell : \mathbb{R}^d \to \mathbb{R}$. In MAS, optimization can be beneficial for achieving efficient and effective outcomes. An unconstrained optimization problem is formulated as:

$$\min_{z \in \mathbb{R}^d} \ell(z) \tag{1.6}$$

where $\ell(z)$ represents the cost function to be minimized, and z denotes the decision vector being optimized. The necessary conditions for optimality in unconstrained problems are:

Necessary Conditions of Optimality:

1. If z^* is a local minimum, then $\nabla \ell(z^*) = 0$.
2. If z^* is a local minimum and $\ell \in C^2$, then $\nabla^2 \ell(z^*)$ is positive semi-definite.

These conditions ensure appropriate behavior of the first and second-order derivatives at the minimum point. Convexity property further simplifies finding optimal solutions. A function $\ell(z)$ is said to be convex if for all $z_A, z_B \in \mathbb{R}^d$ and $\theta \in [0, 1]$, the following holds:

$$\ell(\theta z_A + (1 - \theta) z_B) \leq \theta \ell(z_A) + (1 - \theta) \ell(z_B). \tag{1.7}$$

Convexity is crucial as it ensures global solutions. This importance is underscored by the following propositions from Nonlinear Programming book by Dimitri Bertsekas:

Proposition (Prop. B.10 Nonlin. Progr.): Let $Z \subset \mathbb{R}^d$ be a convex set and $\ell : Z \to \mathbb{R}$ a convex function. Then a local minimum of ℓ is also a global minimum.

Proposition(Prop. 1.1.2 Nonlin. Progr.): Let $\ell : \mathbb{R}^d \to \mathbb{R}$ be a convex function. Then z^* is a global minimum if and only if $\nabla \ell(z^*) = 0$.

For the unconstrained minimization of a convex function, the first-order necessary condition of optimality is also sufficient, and the gradient is a monotone operator:

$$(\nabla \ell(z_A) - \nabla \ell(z_B))^T (z_A - z_B) \geq 0. \tag{1.8}$$

Convexity and gradient monotonicity are pivotal in designing gradient-based optimization algorithms. They ensure that iterative methods, such as Gradient Descent, reliably guide the decision vector towards global minima by following the steepest descent path. This systematic update process accellerates convergence and enhances the algorithm's efficiency in solving complex optimization problems. The iterative update structure of GD follows:

$$z^{k+1} = z^k - \alpha^k \nabla \ell(z^k) \tag{1.9}$$

where $\alpha^k > 0$ is the step size, sufficiently small to always ensure a decrease in the cost:

$$\ell(z^{k+1}) < \ell(z^k). \tag{1.10}$$

With a constant step size, every limit point z^* of $\{z^k\}$ is a stationary point, i.e., $\nabla \ell(z^*) = 0$.

1.1.3 Gradient Tracking Algorithm

Gradient tracking is an advanced optimization technique used to solve the distributed optimization problem. It extends the concept of dynamic average consensus, ensuring that local agents can collectively both track a global gradient estimate over time and lead their own states to optimal solutions, despite having only local information. Dynamic average consensus allows agents to track the average of time-varying signals. Specifically, each agent i maintains an estimate s_i^k of the average signal $\bar{r}^k = \frac{1}{N}\sum_{i=1}^{N} r_i^k$ and runs the consensus update, which consists of an averaging system update plus a local innovation:

$$s_i^{k+1} = \sum_{j\in\mathcal{N}_i} a_{ij} s_j^k + \left(r_i^{k+1} - r_i^k\right) \tag{1.11}$$

where $s_i^0 = r_i^0$. This ensures that the local estimates s_i^k asymptotically track the average signal with bounded error. Building on this concept, gradient tracking updates the local gradient estimates to track the average gradient $\nabla\ell(z^k) = \frac{1}{N}\sum_{i=1}^{N}\nabla\ell(z_i^k)$. Each agent i tracks the gradient of its local cost function $\nabla\ell_i(z_i^k)$ and updates its estimate as follows:

$$s_i^{k+1} = \sum_{j\in\mathcal{N}_i} a_{ij} s_j^k + \left(\nabla\ell_i(z_i^{k+1}) - \nabla\ell_i(z_i^k)\right) \tag{1.12}$$

with the initial condition $s_i^0 = \nabla\ell_i(z_i^0)$. The local descent direction is then taken as the steepest descent, i.e., the negative of the average gradient estimate: $d_i^k = -s_i^k$. Gradient Tracking algorithm updates the decision vector z_i^k and the gradient estimate s_i^k iteratively:

$$\begin{aligned} z_i^{k+1} &= \sum_{j\in\mathcal{N}_i} a_{ij} z_j^k - \alpha s_i^k \\ s_i^{k+1} &= \sum_{j\in\mathcal{N}_i} a_{ij} s_j^k + \left(\nabla\ell_i(z_i^{k+1}) - \nabla\ell_i(z_i^k)\right) \end{aligned} \tag{1.13}$$

where $z_i^0 \in \mathbb{R}^d$ and $s_i^0 = \nabla\ell_i(z_i^0)$. Convergence of the algorithm relies on two assumptions:

1. Graph G is connected with doubly stochastic adjacency matrix A.
2. Each cost function ℓ_i is μ-strongly convex and Lipschitz continuous.

Theorem (Gradient Tracking Convergence) Let Assumptions 1 and 2 hold. Then, in cost-couples problems, there exists $\alpha^* > 0$ such that for all $\alpha \in (0, \alpha^*)$ the sequences $\{z_i^k\}_{k\in\mathbb{N}}$ generated by Gradient Tracking converge to the unique consensual solution z^*.

$$\lim_{k\to\infty} \|z_i^k - z^*\| = 0 \quad \forall i = 1, \ldots, N. \tag{1.14}$$

Moreover, the convergence rate is linear, meaning $\forall\ i = 1, \ldots, N$, $\exists\ \rho \in (0, 1)$ holds:

$$\|z_i^k - z^*\| \leq \rho\|z_i^{k-1} - z^*\| \leq \rho^k\|z_i^0 - z^*\| \quad \forall k \in \mathbb{N}. \tag{1.15}$$

In this project, the optimization problem is not cost-coupled, meaning that each agent minimizes its own local cost function. As a result, while Gradient Tracking is used to achieve consensus over time-varying quantities, such as the average gradient and formation's barycenter, the optimal solutions z_i^* will differ across agents in the graph network.

1.1.4 Distributed Aggregative Algorithm

Aggregative optimization is essential in distributed systems, allowing agents to collectively optimize global objectives while managing local targets. Agents independently minimize local decision variables, but since the cost function involves both local and global terms, dynamic average consensus is required for the shared quantities. The gradient tracking algorithm estimates these aggregative quantities, ensuring each agent reaches its own optimal solution. Consider robots in the plane aiming to optimize their positions $z_i \in \mathbb{R}^2$, $i = 1, \dots, N$, through the minimization of a local cost function $\ell_i(z_i, \sigma(z))$. Here, $b \in \mathbb{R}^2$ represents a global target, $r_i \in \mathbb{R}^2$ represents the local target for robot i, and $\sigma(z) = \frac{1}{N}\sum_{i=1}^{N} z_i$ denotes the robots' barycenter. The optimization problem is formulated as:

$$\min_{z_1,\dots,z_N} \sum_{i=1}^{N} \ell_i(z_i, \sigma(z)), \quad \text{with} \quad \sigma(z) = \frac{1}{N}\sum_{i=1}^{N} \phi_i(z_i) \tag{1.16}$$

The problem involves two global terms: $\sum_{j=1}^{N} \frac{\partial}{\partial \sigma}\ell_j(z_j, \sigma)$ and $\sigma = \frac{1}{N}\sum_{j=1}^{N}\phi_j\left(z_j^k\right)$, respectively the cumulative gradient of the cost function with respect to $\sigma(z)$ and $\sigma(z)$ itself. Each agent i is aware only of ϕ_i and ℓ_i, and maintains estimates z_i^k of $z_i^\star$ and trackers s_i^k of $\sigma\left(z^k\right)$, and v_i^k of $\sum_{j=1}^{N}\nabla_2\ell_j\left(z_j^k, \sigma\left(z^k\right)\right)$. Distributed aggregative algorithm reads:

$$\begin{aligned} z_i^{k+1} &= z_i^k - \alpha\left(\nabla_1\ell_i\left(z_i^k, s_i^k\right) + \nabla\phi_i\left(z_i^k\right)v_i^k\right) \\ s_i^{k+1} &= \sum_{j\in\mathcal{N}_i} a_{ij}s_j^k + \phi_i\left(z_i^{k+1}\right) - \phi_i\left(z_i^k\right) \\ v_i^{k+1} &= \sum_{j\in\mathcal{N}_i} a_{ij}v_j^k + \nabla_2\ell_i\left(z_i^{k+1}, s_i^{k+1}\right) - \nabla_2\ell_i\left(z_i^k, s_i^k\right) \end{aligned} \tag{1.17}$$

with initial conditions given as: $z_i^0 \in \mathbb{R}^{n_i}$, $s_i^0 = \phi_i\left(z_i^0\right)$, and $v_i^0 = \nabla_2\ell_i\left(z_i^0, s_i^0\right)$.

1.2 Problem Set-Up

1.2.1 Graph Representation

The network among robots is represented using a connected graph, with nodes for robot and edges for communication links. The adjacency matrix `Adj` specifies direct connections between robots, and Metropolis-Hastings weights `W` are assigned to edges based on the network structure, determining the influence of neighboring robots during optimization.

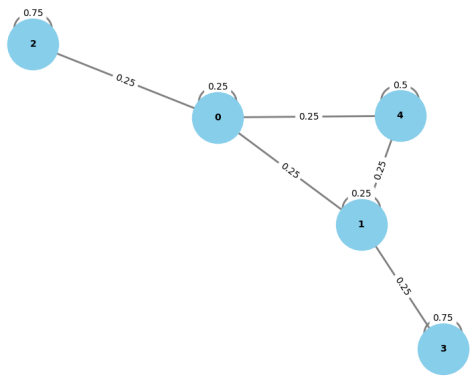


Figure 1.1: Graph Connectivity, 5 agents

1.2.2 Cost function

The method's effectiveness hinges on the cost's structure, which dictates agent's behavior:

$$\ell_i\left(z_i, \sigma(z)\right) = \gamma_{r_{lt}} \left\|z_i - r_i\right\|^2 + \gamma_{bar} \left\|\sigma(z) - b\right\|^2 + \gamma_{agg} \left\|z_i - \sigma(z)\right\|^2 + obst\_pot \tag{1.18}$$

which is made of different components, each of them influencing the cost function:

- **Local Target Attraction Term** ($\gamma_{r_{lt}} \times d_{robot\_target}$), based on the distance of each robot to its local target and weighted by $\gamma_{r_{lt}}$;
- **Barycenter Goal Term** ($\gamma_{\text{bar}} \times d_{\text{robot\_barycenter}}$), based on the distance between the barycenter and global target, weighted by γ_{agg};
- **Barycenter Attraction Term** ($\gamma_{\text{agg}} \times d_{\text{barycenter\_target}}$), based on the distance between the robot and the formation's barycenter.
- **Potential Function**, based on the distance between the robot and each obstacle point, weighted by K, treated in subsection 1.2.3;

The value of the cost `F` is stored to monitor the algorithm's performance, together with the gradient of the cost with respect to the barycenter's position V, and the barycenter itself S (trackers). Those quantities are iteratively tracked by the algorithm over the iterations.

1.2.3 Obstacle Avoidance

The potential fields method enables robots to dynamically adjust their paths in high-dimensional environments to avoid collisions when obstacles are detected. The potential function is defined as $U : \mathbb{R}^m \to \mathbb{R}$ and the gradient of U with respect to q is given by:

$$\nabla U(q) = \begin{pmatrix} \frac{\partial U(q)}{\partial q_1} \\ \vdots \\ \frac{\partial U(q)}{\partial q_n} \end{pmatrix}$$

In subsequent chapters, this method will be presented for obstacle avoidance within the house environment. Obstacles are discretized into finite coordinate points, each linked to a repulsive potential field, indicating unsafe areas for the robots' paths. This ensures robust and adaptive obstacle handling, crucial for safe and efficient navigation. The repulsive force increases as the robot approaches an obstacle, starting from a designated safety distance. The potential function U_{rep} and its gradient used in this project are defined as:

$$U_{\text{rep},i}(q) = \begin{cases} \frac{1}{2} K_{r_i} \left(\frac{1}{d_i(q)} - \frac{1}{q^*}\right)^2 & \text{if } d_i(q) < q^* \\ 0 & \text{otherwise} \end{cases}$$

$$\nabla U_{\text{rep},i}(q) = \begin{cases} K_{r_i} \left(\frac{1}{q^*} - \frac{1}{d_i(q)}\right) \frac{\nabla d_i(q)}{d_i(q)^2} & \text{if } d_i(q) \leq q^* \\ 0 & \text{if } d_i(q) > q^* \end{cases}$$

This method guides robots toward their goals while avoiding collisions. However, a significant issue with potential fields is the occurrence of local minima, where a robot can become trapped in a position that is neither at the goal nor moving towards it. Future developments aim to overcome these limitations by engineering more advanced algorithms.

Chapter 2

Environment

The Environment class is crucial in a multi-agent system (MAS), shaping the physical and dynamic landscape in which agents can operate. Keeping the environment class separate from agent class enhances modularity, maintainability, and scalability of the overall system.

2.1 House Layout

The Environment class constructs the house layout, defining the operational space for agents. The house includes rooms such as *Kitchen*, *Room*, *Office*, *Living Room*, *Corridor*, *Entrance*, and *Exit*, with parameters loaded from *MAS_Project_Parameters.yaml*. Each room's coordinates are scaled to maintain proportional dimensions and stored in a dictionary for easy access and modification. Walls, represented as segments between coordinates, establish boundaries and internal divisions, including openings for entrance and exit. These walls ensure clear navigation, while openings facilitate movement within the house. Their coordinates are scaled and stored for dynamic adjustments. Midpoints aid navigation by serving as intermediate targets in paths between rooms. Obstacles, based on wall coordinates and openings, are discretized into points using a specified step size. The number of points varies with the discretization factor, providing detailed boundaries. Entrance obstacle points are generated perpendicular to the entrance for safe navigation. These artefacts create a structured landscape for agent interaction in the MAS. Each obstacle point is associated with a repulsive potential function, generating a repulsive force to prevent robots from getting too close. This technology, previously discussed in section 1.2.3, requires balancing the number of points to manage computational burden effectively.

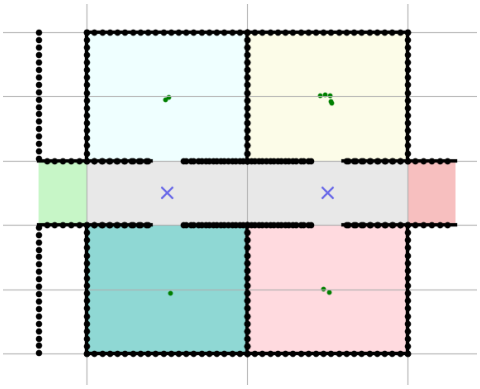
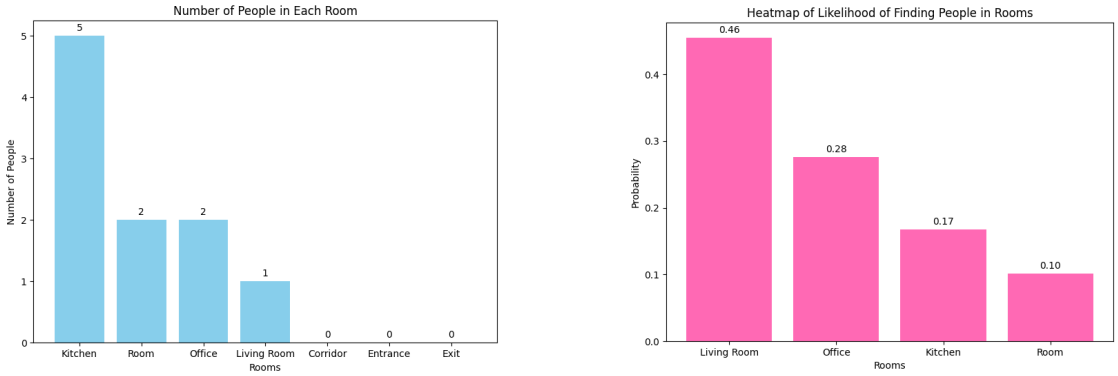


Figure 2.1: House Layout With Visible Obstacles

2.2 Heatmap Generation and People Initialization

People in the environment are modeled using a heat-map that defines the likelihood of finding individuals in each room, guiding agents in prioritizing search areas. The heat-map is created with a weighted probability method, assigning probabilities based on proximity to a central reference room, with exponentially decaying weights normalized to sum to one. The process starts by randomly shuffling the rooms and selecting a central one, which represents the mean of the distribution. Probabilities are assigned based on distance from this room, allowing agents to make informed search decisions without knowing the exact number of people in each venue. Once the heat-map is generated, survivors are initialized based on these probabilities. The expected number of people per room is calculated, and individuals are assigned positions by ensuring the total number matches the expected count. Positions are generated radially around room centers to simulate natural dispersion.



(a) Number of People in Rooms

(b) Probability of People in Rooms

Figure 2.2: People Initialization Process

Survivors are modeled by a Finite State Machine (FSM), with each state representing a different stage in the survivor's interaction with agents and the environment in the MAS:

- **Steady State**: Survivors remain in their initial positions unless interacted with by an agent, to simulate a waiting-to-be-rescued scenario in the rooms.
- **Escorted State**: Survivors who have been reached by the agents, following them while maintaining a fixed radius around the formation's barycenter.
- **Saved State**: Once escorted to the exit, survivors move towards the ambulance by using a proportional controller. The direction of movement is the vector difference between the survivor's current position and the exit, scaled by a step size.
- **Transported State**: After reaching the ambulance, survivors update their positions to match the vehicle's, by simulating transport to the hospital.
- **Healed State**: Once the ambulance reaches the hospital, survivors flag to indicate successful transport. Therefore, no further actions are required in this stage.

FSM transitions are managed by specific conditions and actions, ensuring dynamic and state-dependent behaviors. This interaction between agents and survivors adds depth to the simulation, testing the agents' navigation and assistance capabilities effectively.

2.3 Ambulance Initialization

The Ambulance is a critical component in the MAS, responsible for transporting rescued individuals to safety. Its operation is modeled using a Finite State Machine (FSM) with states including *Idle*, *Departing*, and *Returning*. The FSM ensures efficient and timely transport of survivors, dynamically responding to the environment's conditions.

- **Idle State**: The ambulance waits at the exit for survivors to be saved. It monitors the status of survivors and transitions to the *Departing* state when there are no more survivors in the *Saved* state and at least one survivor in the *Transported* state.
- **Departing State**: The ambulance moves vertically toward the hospital, remaining in this state until it arrives and updates the survivors' states to healed. Upon reaching the hospital, it spawns a new ambulance at the gate if there are still survivors needing rescue, by ensuring continuous transport for people waiting at the exit.
- **Returning State**: In the *Returning* state, the ambulance waits for a predefined countdown period (i.e., *return_iterations* variable defined in parameters' file) before moving back to the exit. The movement is similar to the *Departing* state, ensuring that the ambulance returns to its initial position. If all survivors have been healed at the hospital, then the ambulance will not return, marking the end of the operation.

The Ambulance class also includes visualization functionalities, plotting the ambulance within the environment. The ambulance is graphically represented with detailed features such as its body, wheels, and a red cross, with blinking lights indicating its active state. This visualization aids in understanding the ambulance's position and state within the environment as well as interactions with external entities. Interaction between the ambulance and other agents is seamlessly managed within the *Environment* class. The ambulance monitors survivors' statuses and adjusts its behavior accordingly. For instance, when a survivor is in the *Transported* state, the ambulance ensures they are moved to the hospital. The ambulance's movements and states are updated in real-time, reflecting the dynamic nature of the simulation and ensuring that the rescue mission is efficiently carried out.

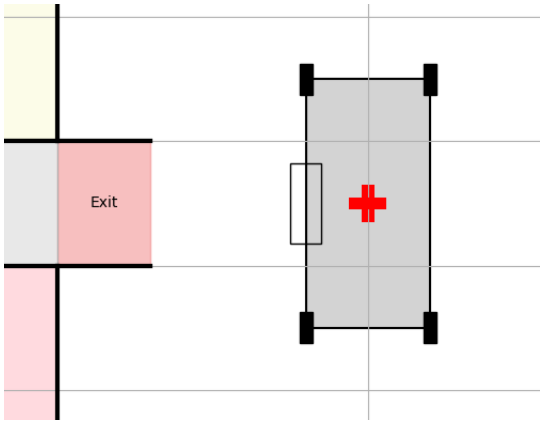


Figure 2.3: Ambulance Visualization

2.4 House Visualization

Visualization is a crucial aspect of the *Environment* class, aiding in understanding the layout and interactions within the environment. The *plot_house()* method is responsible for this, displaying the environment's structure, including walls, rooms, corridors, and entrances/exits. Room midpoints are also plotted to provide clear navigation references.

- **Color-Coded Rooms**: Each room is color-coded to distinguish between different areas in the house with ease.
- **Obstacle Points**: Obstacle points are plotted as black scatter points to visualize potential hindrances in navigation.
- **Midpoints**: Midpoints within rooms are marked with blue crosses, serving as key reference points for navigation between two different rooms.
- **People Positions**: People are depicted as green circles, indicating their starting positions in each room.
- **Entrance and Exit Points**: Marked with green and red symbols respectively, these points highlight entry and exit locations.

This visualization helps to understand the environment's structure and the initial distribution of people in the MAS. It also serves as a valuable tool for debugging and refining the simulation, ensuring that the overall environment behaves as expected.

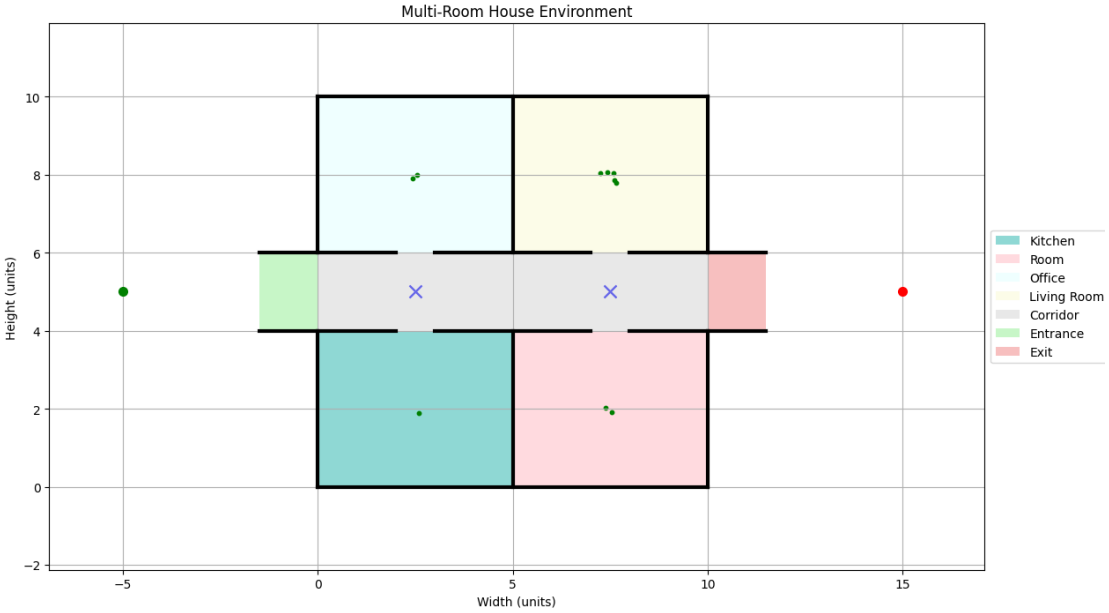


Figure 2.4: House Visualization

Chapter 3

BDI Agents

In a Multi-Agent System, the concept of agency is paramount to effectively model the dynamic interactions of the entities, called agents, in the system. They are autonomous as they are able to effectively perceive their environment, reason about their actions, and make proper decisions to achieve specific goals. The choice for this project is the BDI (Belief-Desire-Intention) framework, a well-established paradigm recognized for its ability to simulate human-like decision-making processes, adopted in various domains due to its strong notion of agency and its capacity to handle complex and dynamic environments.

3.1 BDI Framework

BDI (Belief-Desire-Intention) agents are widely recognized in agent technology for their robust and flexible architecture. Defined by Rao and Georgeff, the BDI framework models the agent's structure through three core components:

- **Beliefs**: Represent the agent's knowledge about the world, including information from perception devices, messages from other agents, and internal data.
- **Desires**: Reflect the states of the world the agent aims to achieve.
- **Intentions**: The plans and actions chosen to fulfill the agent's desires, often represented as post-conditions and goals.

Unlike reactive agents, which respond to stimuli with predefined actions, BDI agents reason and adapt, making them ideal for environments needing strategic planning and flexibility.

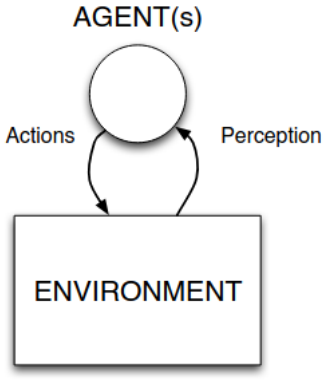


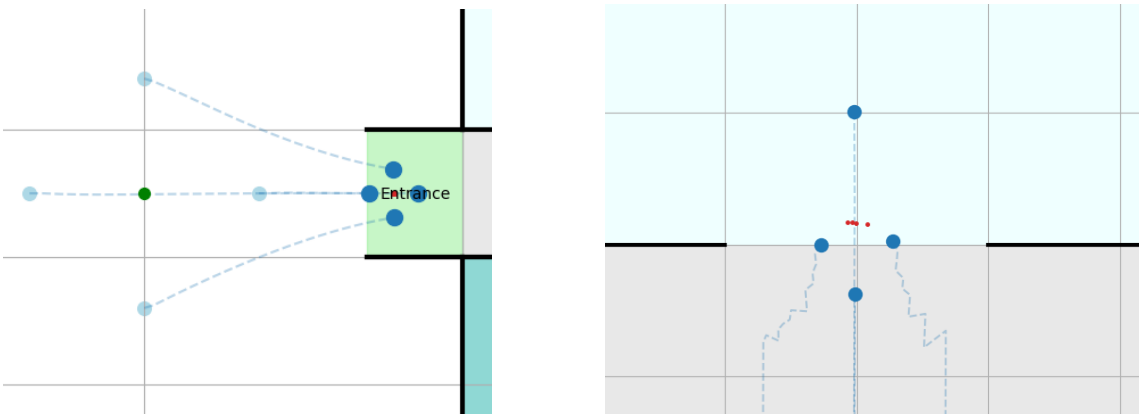
Figure 3.1: Agent(s) Model: Russel and Norvig, 2022

3.1.1 Beliefs

Beliefs represent the agent's knowledge about the environment and its state. In the code, beliefs are initialized in *initialize_beliefs()* function by loading various parameters from the YAML file and the *Environment* class. Such data provide information about ambulance, rooms, obstacles, and the state of other agents in MAS. The structure of beliefs includes:

- **Rooms to Visit**: List of rooms the agent needs to visit, derived from the heat-map.
- **Current Room**: The agent's current room in the house.
- **People Status**: Status of people within the environment (e.g., *Steady*, *Escorted*).
- **People To Escort**: List containing the group of people to be escorted to exit.
- **People Left In Room**: Dictionary with count of survivors remained in each room.
- **Visited Rooms**: List containing the names of visited rooms.
- **Obstacles**: Dictionary containing obstacle points in each room.
- **Midpoints**: Dictionary containing midpoints' coordinates associated to each room.
- **Ambulance State**: The state of the ambulance (e.g., *Idle*, *Departing*, *Returning*).
- **MAXITERS**: Maximum number of iterations for the algorithm, i.e., a deadline.
- **Gains & Radius**: Parameters instructing robots on the formation's shape.

Beliefs are continually updated to reflect the dynamic environment through the *perceive_environment()* and *belief_revision_function()* (BRF). The first function gathers new perceptions, such as the positions of people and the state of the ambulance, ensuring the agent operates with the latest environmental data, mirroring the human sensory process. After gathering perceptions, the BRF updates the agent's beliefs, ensuring its knowledge remains current, akin to how humans update their understanding based on new observations. Dynamic information, like the status of survivors, is frequently updated, while static information, such as obstacle positions, remains unchanged unless explicitly updated. This continuous belief update process is critical for informed decision-making, allowing the agent to adapt its actions effectively and demonstrate high level of autonomy.



(a) Entering The House, Static View

(b) Entering a Room, Dynamic View

Figure 3.2: Belief's Change in Formation's Shape due to Obstacles

3.1.2 Desires

Desires represent the agent's goals, generated based on current beliefs and the environment's state using the *generate_options()* function. This function determines the agent's next objectives by evaluating the environment and generating possible goals. For example, if the agent believes there are people to escort in the current room, it generates the desire to escort them. Desires can be multiple and conflicting. For instance, the agent might want to explore a new room while needing to escort survivors. The function handles this by generating all possible desires and evaluating their feasibility based on beliefs. This ensures the agent can identify and prioritize its goals effectively. Possible desires include:

- **Visit Rooms**: List of rooms yet to be visited, with the first one being the target.
- **Escort People**: Presence of survivors to be escorted in the current room.
- **Standby Mode**: Enter standby mode when all people are saved or transported.
- **Exit House**: Desire to leave the house once all people are saved to end the mission.
- **Should Return**: Returning to the previous room if some people are left there.

The flexibility to manage multiple, conflicting desires allows BDI agents to balance immediate actions with long-term goals. This capability is crucial in complex environments, enabling agents to navigate unpredictability and mimic human-like decision-making.

3.1.3 Intentions

Intentions are the plans and actions that the agent commits to in order to achieve its desires. The *filter_options()* function is used to prioritize and select the most appropriate intentions based on the current desires and beliefs. Unlike desires, only one desire can be transformed into an intention at a time. This function inherently prioritizes the desires, ensuring that the most critical goal is addressed first. The structure of intentions includes:

- **Target Room**: The name of the room agents plan to visit next.
- **Target Position**: The center's coordinates of the room agents plan to visit next.
- **Escort People**: If agents should escort people in the current room, this intention sets the target room to *Exit* and updates the target coordinates accordingly.
- **Exit**: Intention to exit the house when all people are saved or the mission is complete.
- **Standby Mode**: Entering standby mode when all people are saved or transported.
- **Explore House**: Explore unexplored rooms in beliefs' *rooms_to_visit* list.
- **Formation Parameters**: Parameters like *gain*, *radius*, *targets_attraction*, *barycenter_attraction*, and *barycenter_repulsion* shape the distributed formation of robots.

The *filter_options()* function works by assigning priorities to each desire. For instance, escorting people to safety might have a higher priority than exploring a new room. Similarly, returning to the previous room where survivors were left should be more important than proceeding to the next room. Hence, such a function selects the highest-priority desire and transforms it into an intention. This process ensures that each agent focuses on the most important tasks, managing its own resources and actions effectively to reach its goal.

3.2 Plan Creation and Execution

Once the intention has been generated and prioritized, the agent needs to develop and execute a plan to achieve it. This process involves determining the next room to visit, creating a path to reach it, and executing movements in a coordinated and efficient manner.

3.2.1 Plan Creation

The *create_plan()* function generates a plan that the agent follows to achieve its current intention. This function constructs a list of waypoints, which serve as intermediate targets for navigation, and a list of obstacles points to be avoided in the process. Key steps include:

- **Identify the Next Room**: Based on the intention, select the next room from the visit list for exploration or set the target room to the exit for escorting people.
- **Construct Path**: Create a path between the current room and the target room by identifying midpoints and adding them to the rooms' centers coordinates.
- **Assemble Escort Group**: If escorting, gather individuals needing escort from the current room using *create_escort_group()* and update their status to "Escorted."
- **Collect Obstacles**: Identify and include obstacles associated with the current and target rooms in the plan to ensure safe navigation and collision avoidance.

These beliefs provide agents with the knowledge necessary to make informed decisions, select appropriate desires and intentions, and navigate their environment effectively.

3.2.2 Plan Execution

The *execute_plan()* function implements the plan from *create_plan()*. It iteratively advances the agent towards its target using a distributed aggregative optimization algorithm, by maintaining formation and ensuring efficient movements. The execution involves:

- **Targets Handling**: Define next local and global targets (R and b) by checking arrival and updating to complete the plan. Update current room's belief upon arrival.
- **Optimization Problem**: Navigate to targets by solving the optimization problem with the distributed aggregative optimization algorithm. Agents update positions based on local data, maintaining a tight formation and coordinated movements.
- **Communication Actions**: Continuously update agents' positions according to local cost functions, by ensuring consensus on coupling variables with message-passing.
- **Pragmatical Actions**: Agents dynamically act on the environment through pragmatical actions, by changing ambulance and survivors' states acting on their FSMs.
- **Escorting to Exit**: When escorting people to the exit, upon reaching the exit room, update beliefs by removing the escort group and marking survivors as *Saved*.

Execution starts by setting the next target and solving the aggregative optimization problem, then the agents' positions are updated iteratively to reach each waypoint. The *should_optimize()* function checks if the final plan's target is reached or the maximum iterations are exceeded, marking the plan's completeness. The *create_plan()* and *execute_plan()* routines enable agents to navigate through the environment, avoid obstacles, and fulfill the SAR's mission objectives. This structured approach ensures the agent's actions are coherent, goal-directed, and adaptable to the environment's dynamic nature.

3.3 Control Loop

The control loop of BDI agents represents their "minds," continuously updating beliefs, generating desires, filtering options into intentions, and executing plans. It ensures that agents dynamically respond to environmental changes, update knowledge, and adjust actions to achieve the goals. Project's implementation is shown in the following pseudo-code:

```
while True:
    perceptions = perceive_environment()
    beliefs = belief_revision_function(perceptions, beliefs)
    desires = generate_options(beliefs)
    intentions = filter_options(beliefs, desires)
    plan, obstacles = create_plan(beliefs, intentions)
    execute_plan(plan, obstacles, beliefs, desires, intentions)
    if check_termination(): break
```

Each step of the BDI loop plays a role in the overall agent's decision-making process:

- **Perceive Environment**: Gathers the latest data about the environment, updating the agent's knowledge and ensuring agents' awareness of their surroundings.
- **Belief Revision Function**: Updates agents' beliefs to reflect new information in the MAS, ensuring decisions are based on the current state of the environment.
- **Generate Options**: Creates possible goals based on the updated beliefs. This step explores various actions the agent can take, considering the current context.
- **Filter Options**: Prioritizes and selects the most relevant intention from the generated options, by evaluating the feasibility and importance of each available desire.
- **Create Plan**: Develops a step-by-step plan to achieve the selected intentions. It identifies the necessary actions and waypoints, considering obstacles in the path.
- **Execute Plan**: Carries out the plan, directing the agents towards their goals. This step involves solving the optimization problem and updating robots' positions.
- **Check Termination**: Evaluates if the mission is complete or the loop should continue, ensuring the agent stops only when goals are achieved or conditions are met.

The loop ensures agents adaptively interact with the environment, making decisions that dynamically move them closer to their own goals. It represents the core of their decision-making process, embodying the BDI framework's focus on adaptation and rational action.

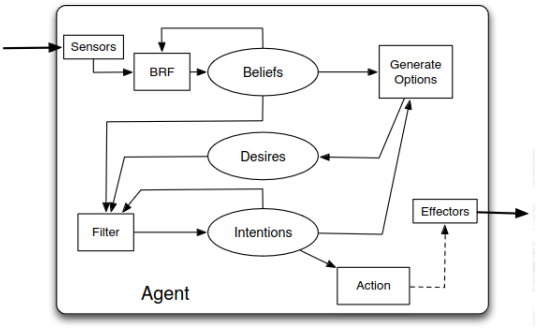


Figure 3.3: Basic Architecture of a BDI Agent, Wooldridge 2009

Chapter 4

Search and Rescue Operations

4.1 Trajectory Initialization

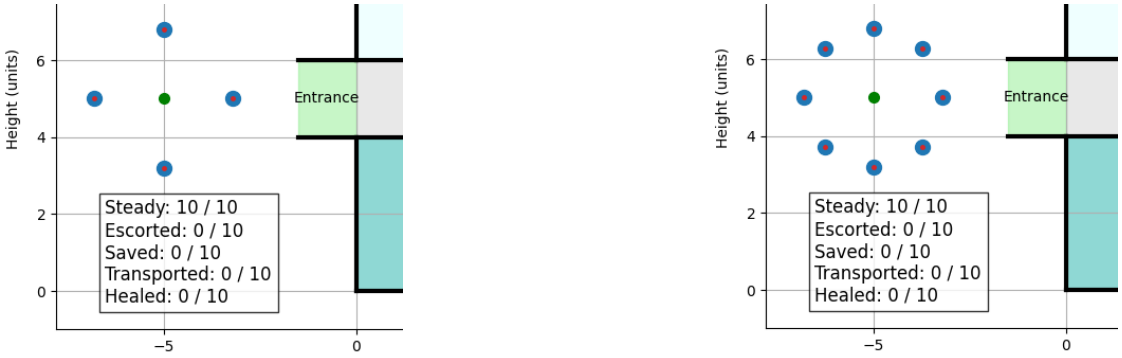
Trajectory initialization is the first step of SAR operations, defining the trajectories for robots, people, and the ambulance, in order to enable effective asynchronous visualization.

4.1.1 Circular Formation Initialization

Robots are initially deployed in a circular formation around the entrance point. The *generate_circular_positions()* function calculates their positions by dividing the circle into equal segments using trigonometric methods. Robots' trajectories are multi-dimensional, spanning iterations, agents, and space, thus enabling detailed tracking and visualization.

- **Number of Agents (N)**: Total robots to be deployed.
- **Dimensions (n_z)**: Robots position 2D coordinates.
- **Central Position**: The focal point of the formation, initially the entrance.
- **Radius**: Determines the circle's size.
- **Gain Factor**: Adjusts the formation's density by changing the distance from point.

The result of the function is a balanced, evenly spaced initial deployment that prepares the robotic agents for effective navigation in the house, as shown in figure 4.1.



(a) Robots Formation, 4 Agents

(b) Robots Formation, 8 Agents

Figure 4.1: Robots Trajectories Initialization

4.1.2 Trajectories Initialization

The *trajectories_initialization()* function sets up the movement paths for agents, the ambulance, and survivors. It also defines targets, initializes position arrays, and prepares monitoring systems crucial for mission success. Trajectories are structured across iterations and space, with robot trajectories also incorporating the agent dimension. This setup allows efficient data collection, allowing for asynchronous post-operation visualization without requiring real-time processing. Key components of the function include:

- **Global Variable Setup**: Initializes iteration counters and state variables like *next_room_name*, *room_reached*, and *people_escorted*.
- **Trajectories Initialization**: Prepares trajectories for local and global targets (R and b), robot positions (Z), barycenter estimates (S), and cumulative gradients (V).
- **Targets Definition**: Utilizes *generate_circular_positions()* to set the robots' starting and final target positions, i.e., local (R) and global (b) coordinate objectives.
- **Cost and Gradient Monitoring**: Initializes arrays to track the cost function (F) and gradients (*grad_z* and *grad_s*), essential for optimizing movements.

This thorough initialization ensures that each robot is optimally positioned and ready to follow a structured path, facilitating efficient and effective task execution.

4.2 Techniques and Procedures

SAR operations are executed through coordinated steps: entering the house, searching for survivors, escorting them to the exit, and returning to rooms where survivors remain. This process ensures all areas are searched and all survivors safely escorted to the exit.

4.2.1 Entering The House

Upon entering, agents move towards their designated radial targets positioned in the house entrance, by avoiding obstacles using the potential field method treated in section 1.2.3.

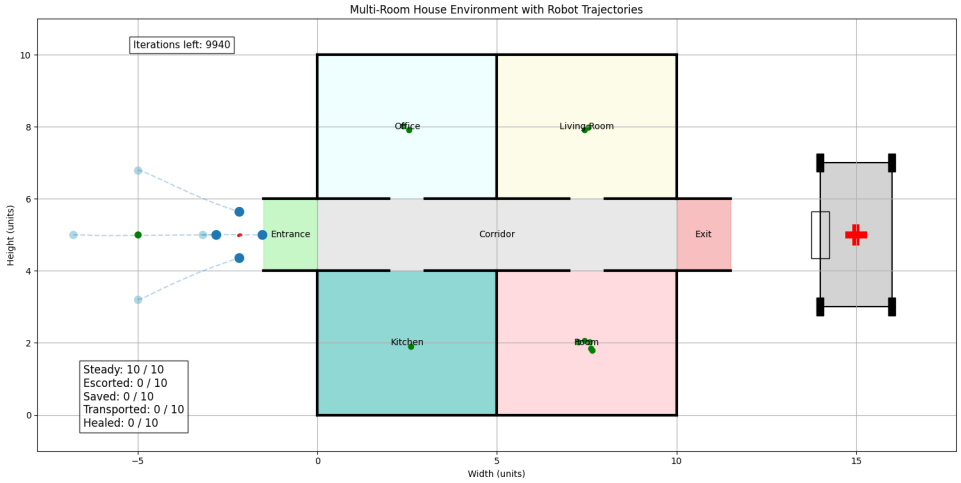
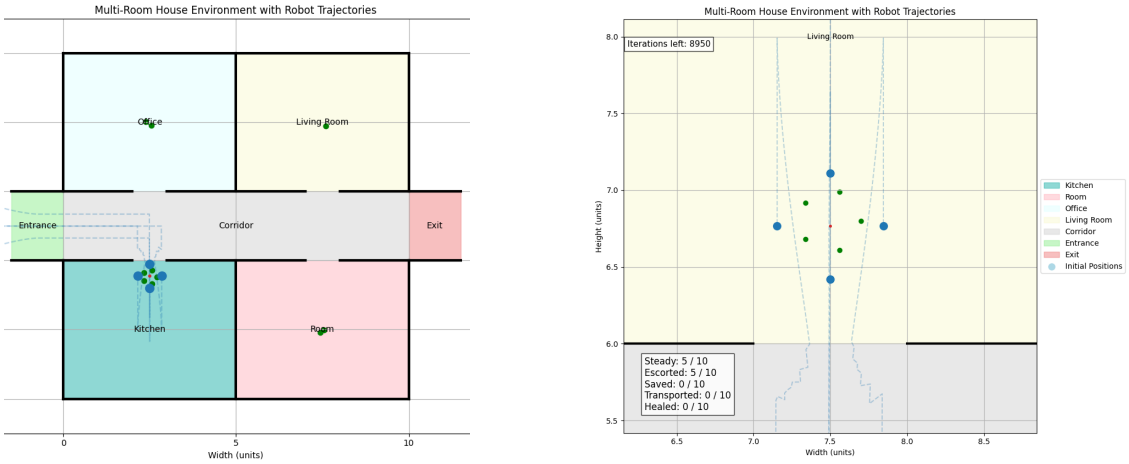


Figure 4.2: Robots Entering The House

4.2.2 Escorting Survivors

In SAR missions, agents are tasked with systematically searching for survivors, escorting them to safety, and returning to any rooms where survivors may have been left due to capacity limitations. The agents utilize their sensory capabilities to detect survivors within each room, by updating their beliefs accordingly. If survivors are found in the *Steady* state, the agents prepare to escort them. The *max_survivors_escort* parameter dictates the maximum number of survivors that can be escorted in a single operation. Hence, when the number of survivors in a room exceeds this threshold, agents may leave some survivors behind to be escorted later. Upon determining that survivors need to be escorted, agents utilize the *create_escort_group()* function to assemble the escort group from the current room. The agents then navigate towards the exit, by maintaining formation and ensuring that all individuals in the group are safely guided to the designated exit point. The movement is carefully managed through formation parameters, such as gain and radius, which are dynamically adjusted based on the environment, ensuring that the group moves efficiently towards the exit. The agents' use of midpoints and specific room coordinates facilitates efficient navigation, ensuring that no house's rooms are missed and that all survivors are accounted for. In situations where survivors are left behind in previous rooms due to the *max_survivors_escort* limit, the agents are programmed to prioritize returning to these rooms. This intention to return is handled by the prioritization mechanism within the *filter_options()* function, which ensures that returning to rescue the remaining survivors takes precedence over exploring new rooms. This mechanism is critical for ensuring the completeness and thoroughness of the SAR mission, as it addresses the need to revisit rooms where survivors were initially left behind. This process ensures that all areas of the operational environment are thoroughly searched, all survivors are escorted to safety, and no survivors are left behind unintentionally. Figures 4.9a and 4.9b illustrate an example of an escorting operation, by providing both static and dynamic views of the simulation.



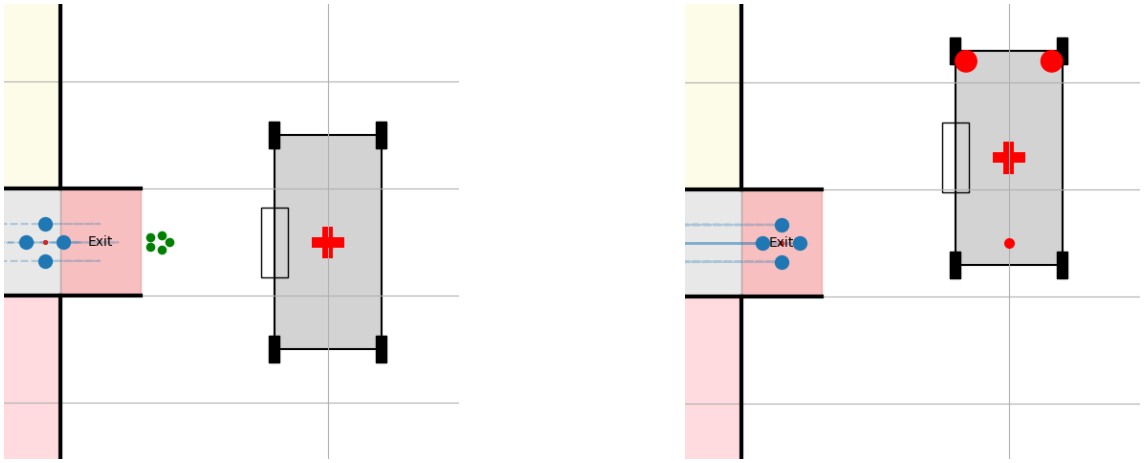
(a) Escorting People, Static View

(b) Escorting People, Dynamic View

Figure 4.3: Escorting People in SAR Mission

4.2.3 Ambulance Movements

Throughout SAR operations, the ambulance transports rescued individuals to safety, closely coordinated with the agents' actions for timely and efficient survivor transport. Governed by a Finite State Machine (FSM) with states like *Idle*, *Departing*, and *Returning*, the ambulance dynamically adapts to the environment and agents' needs, enhancing mission efficiency by adjusting its state based on survivor detection and mission progress.

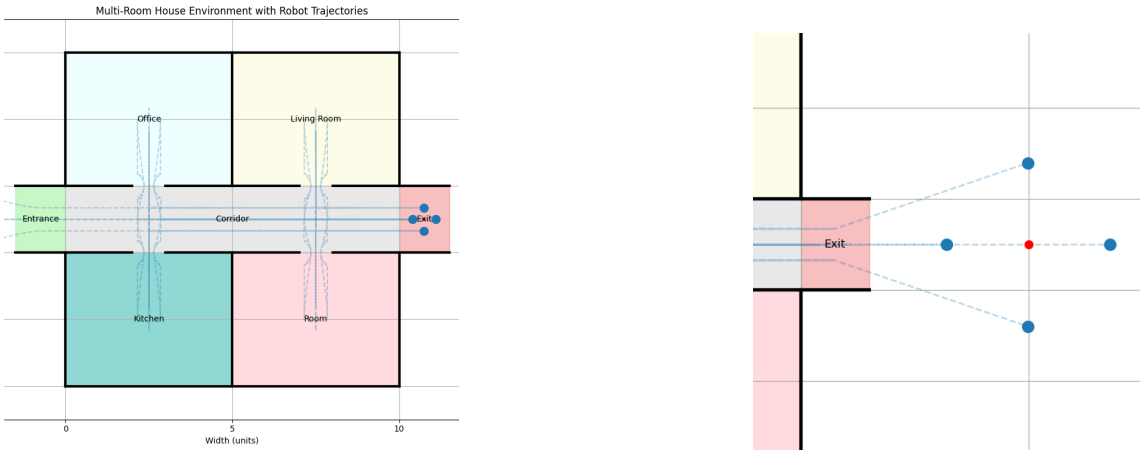


(a) Ambulance in Idle State (b) Ambulance in Departing State

Figure 4.4: Ambulance Visualization in SAR Mission

4.2.4 Exiting the House

After escorting all survivors to the exit, agents may enter standby mode within the exit room whenever some survivors remain unhealed. In this mode, agents must stay alert and continuously monitoring the environment until the completeness. Once all survivors are healed and the area is secure, agents transition to the final phase: exiting the house. The *exit* intention guides the path toward the exit, focusing on efficient movement, obstacle avoidance, and maintaining formation, with an adjusted gain to cover a broader area.



(a) Exiting House, Static View (b) Exiting House, Dynamic View

Figure 4.5: End of SAR Operation: Agent Exit in Dynamic and Static Views

4.3 Visualization

This section covers the techniques used to visualize the MAS' dynamics, presenting plots of costs and gradients used to assess the aggregative optimization algorithm's effectiveness.

4.3.1 SAR Animation

The *SAR_animation()* function visualizes SAR operations by animating the trajectories of robots, survivors, and the ambulance across iterations. Controlled by the *view_type* parameter in YAML file, it provides either a static, top-down view of the house or a dynamic tracking view that follows the robots closely. The function has several key aspects:

- **Plot Limits**: The plot limits ensure consistent visibility of the environment. Based on *view_type* parameter, they can be either static, providing a complete overview of the house environment, or dynamic, adjusting the focus to active agents' regions.
- **House Layout and Robot Trajectories**: The house layout, including walls, color-coded rooms, obstacle points, and corridors, is clearly depicted, with robots' initial positions and trajectories plotted to illustrate their movement strategies in the MAS.
- **Survivors and Ambulance Trajectories**: The movements of survivors and the ambulance are animated to highlight their critical roles in the MAS. Survivors' positions are dynamically updated, showing their progress from detection to rescue. The ambulance, with details like blinking lights, visually reflects its current state.
- **Dynamic Annotations**: Real-time updates on key metrics, like iterations left and survivor status, are provided through dynamic annotations to ensure continuous mission monitoring and insights into agent performance during SAR operations.

The detailed visualization produced by the *SAR_animation()* function allows for in-depth analysis of the SAR mission, making it an essential tool for understanding agent behaviors.

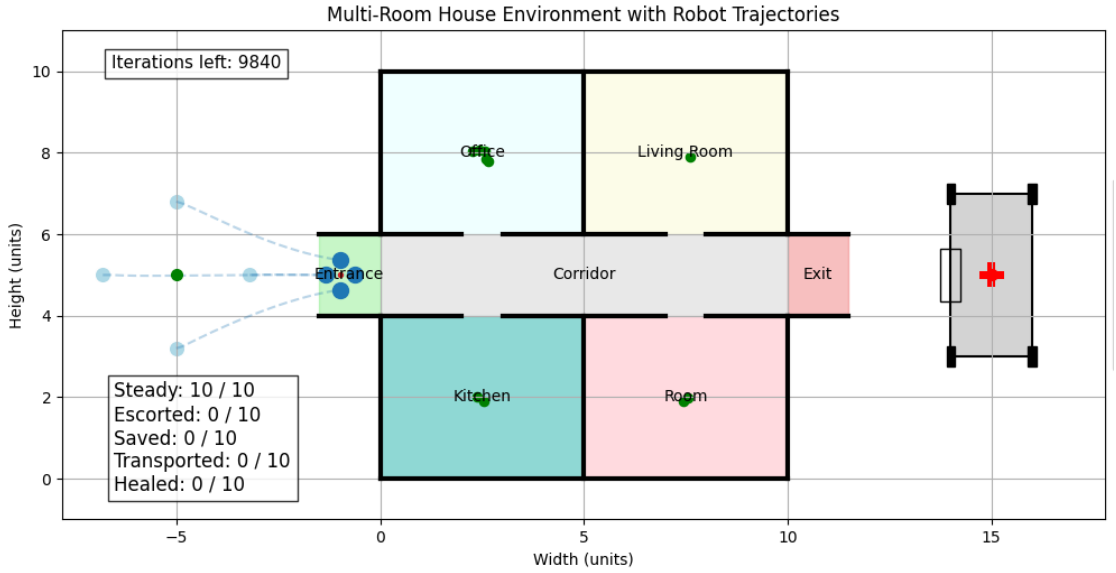
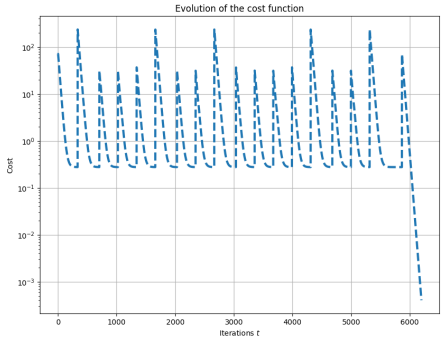


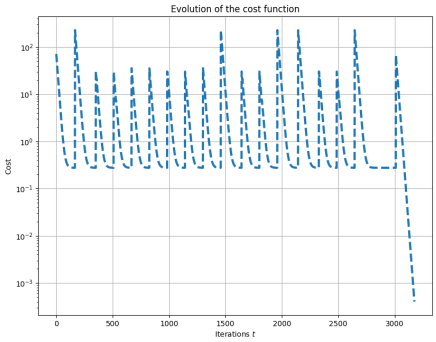
Figure 4.6: Visualization of SAR Operations in the MAS

4.3.2 Cost and Gradients

The *SAR_plots()* function complements the animation by plotting the cost function and gradient norms in a semi-logarithmic scale. These metrics evaluate the distributed aggregative optimization, showing the minimization of local cost functions and the convergence of agents' positions and the team's barycenter. These plots provide key insights into the efficiency of the optimization process and the agents' ability to achieve their own goals.

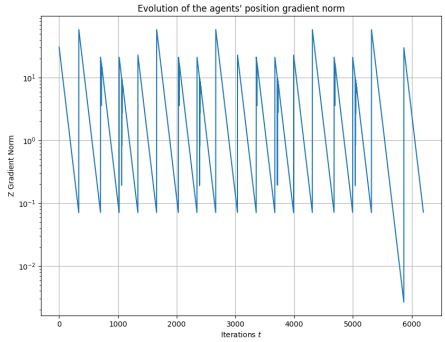


(a) Cost, $\alpha = 1e-2$

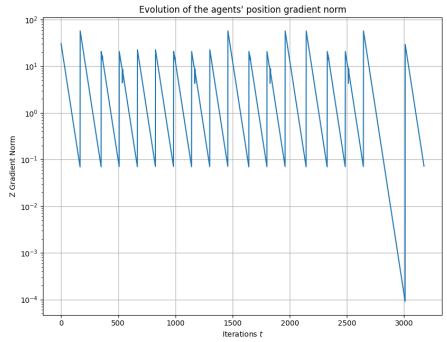


(b) Cost, $\alpha = 2e-2$

Figure 4.7: Cost Function Over Iterations

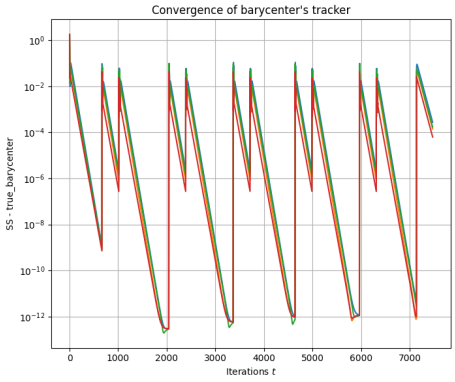


(a) Z Gradient Norm, $\alpha = 1e-2$

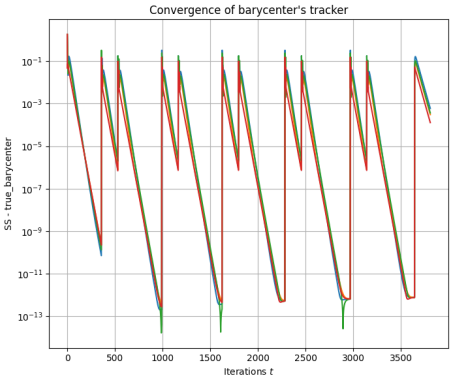


(b) Z Gradient Norm, $\alpha = 2e-2$

Figure 4.8: Evolution of Z Gradient Norm



(a) S - true_barycenter, $\alpha = 1e-2$



(b) S - true_barycenter, $\alpha = 2e-2$

Figure 4.9: Evolution of Barycenter Tracking Error

Conclusions

This report has outlined the design and implementation of a modular Multi-Agent System (MAS) specifically tailored for Search and Rescue (SAR) missions. The system's architecture is built on a clear separation between agents and the environment, ensuring flexibility, scalability, and efficient coordination in complex, dynamic scenarios. Governed by the Belief-Desire-Intention (BDI) framework, the agents autonomously navigate their environment, adjusting their intentions in real time based on updates to their beliefs and desires. The integration of consensus and optimization theories, through the distributed aggregative optimization algorithm, allows agents to achieve mission goals effectively while avoiding obstacles and maintaining a cohesive formation. The modular approach to system design ensured adaptability, facilitating the integration of new agent behaviors, changing room configurations, and evolving mission requirements. This adaptability is critical in SAR operations, where both the environment and mission parameters can shift rapidly. The simulation emphasized task execution, path optimization, and formation control, highlighting the MAS's potential for deployment in more unpredictable, real-world settings. The use of a distributed optimization algorithm proved essential for enabling decentralized, coordinated decision-making, enhancing the agents' autonomy and overall system efficiency in executing SAR operations. Looking ahead, several enhancements could further expand the system's capabilities. By integrating the MAS into the ROS2 environment, the system can transition into real-world applications, leveraging improved hardware communication and real-time performance. In this framework, Simultaneous Localization and Mapping (SLAM) could be implemented to allow agents to explore and construct maps of unknown environments. Introducing battery management with energy consumption constraints would improve mission efficiency, and enhanced health status tracking would enable agents to prioritize tasks more effectively. Multi-Agent Reinforcement Learning (MARL) could increase adaptability in complex environments, while decentralized task allocation (e.g., GA) would optimize resource distribution. These improvements will elevate the overall system's performance, bringing it closer to real-world SAR operations.

Bibliography

- Omicini, A., & Calegari, R. (2024). *M1 - Agents for Complex Distributed Systems* [Slides]. University of Bologna, Multi Agent Systems Course.
- Omicini, A., & Calegari, R. (2024). *M3 - Artefacts for Agents. Function and Use in MAS* [Slides]. University of Bologna, Multi Agent Systems Course.
- Omicini, A., & Calegari, R. (2024). *M5 - Reasoning Agents* [Slides]. University of Bologna, Multi Agent Systems Course.
- Omicini, A., & Calegari, R. (2024). *S1 - Sources of Scientific Literature for Intelligent Autonomous Systems* [Slides]. University of Bologna, Multi Agent Systems Course.
- Omicini, A., & Calegari, R. (2024). *S2 - Systematic Literature Review. A Methodology for Scientific Surveys* [Slides]. University of Bologna, Multi Agent Systems Course.
- Notarstefano, G., & Notarnicola, I. (2024). *Preliminaries on Algebraic Graph Theory* [Slides]. University of Bologna, Distributed Autonomous Systems Course.
- Notarstefano, G., & Notarnicola, I. (2024). *Averaging Systems* [Slides]. University of Bologna, Distributed Autonomous Systems Course.
- Notarstefano, G., & Notarnicola, I. (2024). *Distributed Cost-Coupled Optimization* [Slides]. University of Bologna, Distributed Autonomous Systems Course.
- Notarstefano, G., & Notarnicola, I. (2024). *Distributed Aggregative Optimization* [Slides]. University of Bologna, Distributed Autonomous Systems Course.
- Palli, G. (2023). *Navigation and Localization* [Slides]. University of Bologna, Autonomous and Mobile Robotics Course.
- Russell, S. J., & Norvig, P. (2022). *Artificial Intelligence: A Modern Approach* (4th ed.). Pearson Series on Artificial Intelligence. Pearson Education Limited, Global Edition.
- Bullo, F. (2019). *Lectures on Network Systems* (1st ed.). CreateSpace Independent Publishing Platform.
- Bertsekas, D. P. (2016). *Nonlinear Programming* (3rd ed.). Athena Scientific, Belmont, MA.
- Wooldridge, M. J. (2009). *An Introduction to MultiAgent Systems.* John Wiley Sons Ltd., Chichester, UK, 2nd edition.

- Siciliano, B., Sciavicco, L., Villani, L., & Oriolo, G. (2009). *Robotics: Modelling, Planning and Control*. Springer, London. Advanced Textbooks in Control and Signal Processing.

- Georgeff, M., Pell, B., Pollack, M., Tambe, M., & Wooldridge, M. (1999). *The Belief-Desire-Intention Model of Agency*. In J. P. Mueller, M. P. Singh, & A. S. Rao (Eds.), Proceedings of the 5th International Workshop on Intelligent Agents V: Agent Theories, Architectures, and Languages (ATAL '98), Lecture Notes in Computer Science (Vol. 1555, pp. 1–10). Springer, Berlin, Heidelberg.

- Rao, A. S. and Georgeff, M. P. (1998). *Decision procedures for BDI logics.* Journal of Logic and Computation, 8(3):293–342

- Rao, A. S. and Georgeff, M. P. (1995). *BDI agents: From theory to practice.* In Lesser, V. R. and Gasser, L., editors, 1st International Conference on Multi Agent Systems (ICMAS 1995), pages 312–319, San Francisco, CA, USA. The MIT Press

- Rao, A. S. and Georgeff, M. P. (1992). *An abstract architecture for rational agents.* In Nebel, B., Rich, C., and Swartout, W. R., editors, 3rd International Conference on Principles of Knowledge Representation and Reasoning (KR '92), pages 439–449, Cambridge, MA, USA. Morgan Kaufmann.