Towards a Continous Cleaning Robot

Andrea Gatti[0009−0003−0992−4058]

University of Genoa
andrea.gatti@edu.unige.it

Abstract. Cleaning vacuum cleaner robot is a common didactic example in the field of multi-agent systems. Normally, the robot is placed inside a grid world and reasons directly on the step to perform. In this work, we present a different approach where the robot is placed in a continuous environment and reasons in a more abstract way on regions instead of cells. In this way, the agent is able to abstract more and focus on high-level tasks instead of low-level details.

Keywords: Intelligent Agents · Virtual Environment · Cleaning Robot.

1 Introduction and Motivation

The scenario of the cleaning robot is often used in textbooks as an example to present intelligent agents and their basic principles; one need only cite "Artificial Intelligence: A Modern Approach"[1] by S. Russell and P. Norvig, who use it as early as the second chapter as an introduction to the agent concept. The case study is simple, a robot vacuum cleaner is in a grid environment and has to clean the whole space by having motion and cleaning as actions. There are then various versions related to map knowledge that obviously influence the agent's behavior. In the most basic versions the agent finds out whether a cell contains dirt or not only when it is standing on it, in others it knows where the dirt is on the entire map and must find the best way to complete the task.

The robot vacuum cleaner in these examples reasons about a space being discretized and does so very low-level, deciding what individual steps to take and in what direction. Also, these paths reduce the agents' rotational degrees of freedom to the four directions above, below, right and left. This discretization is certainly necessary if one thinks that an agent moving through the space has to reason not so much about *what* it has to do but more about *how* it has to do it and the individual steps that lead it to achieve its goal. While this may be true for some type of agents, it is certainly reductive for logical agents based on the Belief-Desire-Intention paradigm, which instead have their strength precisely in abstraction and in delegating the management of the environment to external artifacts.

Moreover, in the current state of the art, most simulation software and game engines carry much more advanced path finding technologies than a simplistic discrete representation of the world. Godot[2] implements NavigationPaths, Unity[3] has an entire Navigation System that implements path finding, and

even Unreal Engine[4] implements Navigation Mesh that is used by agents to move through space. All these tools also implement the handling of steps, ascents, descents, and other situations that in a logical implementation should be considered as separate instances and handled properly.

Even in the field of robotic research, we can find multiple approaches to finding optimal paths that consider space in the continuum, finding detailed solutions that do not simply involve left, right steps with predetermined rotations but true paths to be followed with 360-degree freedom of rotation. A complete overview of the state of the art from this point of view can be found in the survey[5] by K. Karur et al. on path planning for mobile robots. The paper makes a first major division between global algorithms in which the agent has complete knowledge of the environment and local algorithms in which, on the other hand, it knows only a neighborhood of its location and must therefore update its path as it moves. The algorithms presented move from discrete to continuous space, demonstrating that here, too, technology has moved beyond a cellular representation of the world.

In the research branch on BDI agents, on the other hand, we find another survey exploring recent spatial reasoning techniques for BDI agents[6]. In the approaches presented, agents reason about where they are, often discretizing, but still always deciding on individual movement directly by the agent.

Some works reason directly about Cartesian coordinates, others about the four directions (front, behind, right, left), and others with structures borrowed from other areas of research (such as Entity-Relation diagrams). Of all the approaches, the most widely used is the Region Connection Calculus, which will be presented in the Design and Implementation section since our implementation also makes use of it.

The paper we present seeks a solution to the problem of robotic vacuuming that takes into account the novelties on the simulation and robotics side, going to use state-of-the-art path planning tools in the continuum, without limiting the robot's degrees of freedom, reasoning about the regions to be explored using a logical agent that makes the implementation furthermore explicable and transparent, as well as enhancing the programming paradigm. In this early work the agent has a comprehensive knowledge of the environment and where the dirt is. An exploratory version of the project is already in the works in which the agent starts without a map and has to create it from scratch. A brief mention of what has been done and future developments will be in the Future Work section.

2 Design and Implementation

To implement an agent that reasons logically about regions and takes advantage of simulation software to do path finding, it is necessary to choose the most suitable tools. Since there is no framework for implementing BDI agents on major game engines, it was necessary to separate the two components and create a connection between them. The first component handles the intelligent agent and the second handles the physical body and environment. The framework must

ensure a 1-to-1 connection between the BDI agent and its body. The architecture can be seen in Figure2.

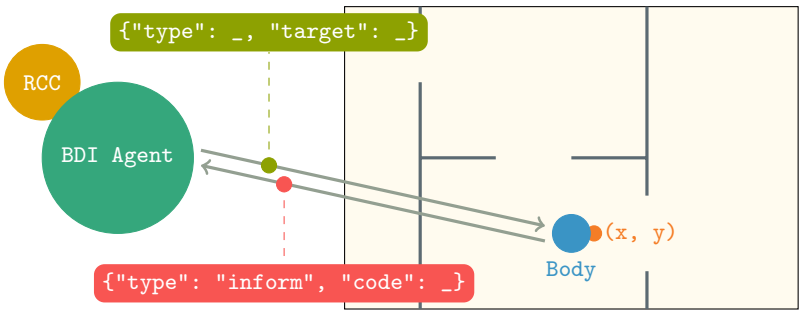

Fig. 1. The architecture of the system

The section is now divided into three subsections. The first lays some fundamentals to be shared by agent and environment on the Region Connection Calculus (RCC) and the environment in which the simulation will take place, the second presents the design and implementation of the BDI agent, and the third follows the same path for the virtual environment.

2.1 Environment and Regions

The environment and agent must properly implement the principles of Region Connection Calculus (RCC) introduced by D. A. Randell, Z. Cui and A. G. Cohn[7]. RCC provides some basic relationships between regions that are visible in Figure2.1. With these relationships it is possible to describe any environment qualitatively, abstracting from coordinates (x, y) and focusing instead on regions. The relations that turn out to be most used within this work are `EC(A, B)`, `PO(A, B)` and `NTPP(A, B)`, which are the essential relations to be able to find an object within a region and know which regions border on which others in order to move between them.

The problem of robot vacuum cleaning is to remove all the dirt from the floor. The relation that carries this information is `NTPP(dirt, Region)` where Region is the region where the dirt is actually contained. We then decided to include another hierarchical concept to make the reasoning finer, adding the concept of a room. Each region is then found contained within a room with the relation `NTPP(Region, Room)`. The rooms are then connected by doors. Doors are considered as regions that have intersection with both one room and the other (`PO(door, Room1)`, `PO(door, Room2)`) but are not contained in either. Only the regions that are traversable by the robot are considered in this initial work without taking into account the objects in the room that are only handled by the simulation side of the framework for the time being. The regions within a room

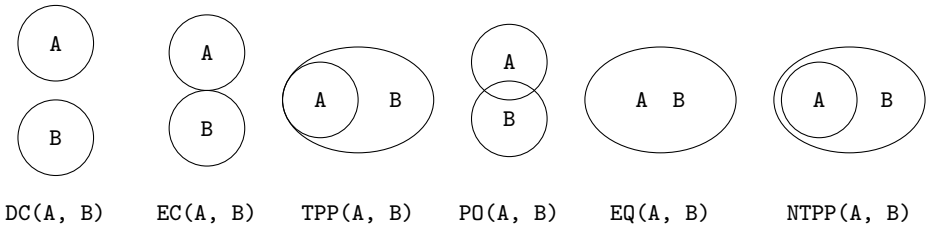

Fig. 2. Relations between regions

are all close but without overlapping, thus respecting the relation `EC(Region1, Region2)`.

In order to give a proof of concept in an environment that was as verisimilar and nontrivial as possible, the study room of computer science PhD students at the University of Genoa was considered as a virtual environment. This environment can be seen in Figure2.1. Using the representation of a real environment makes the simulation nontrivial and verisimilar.

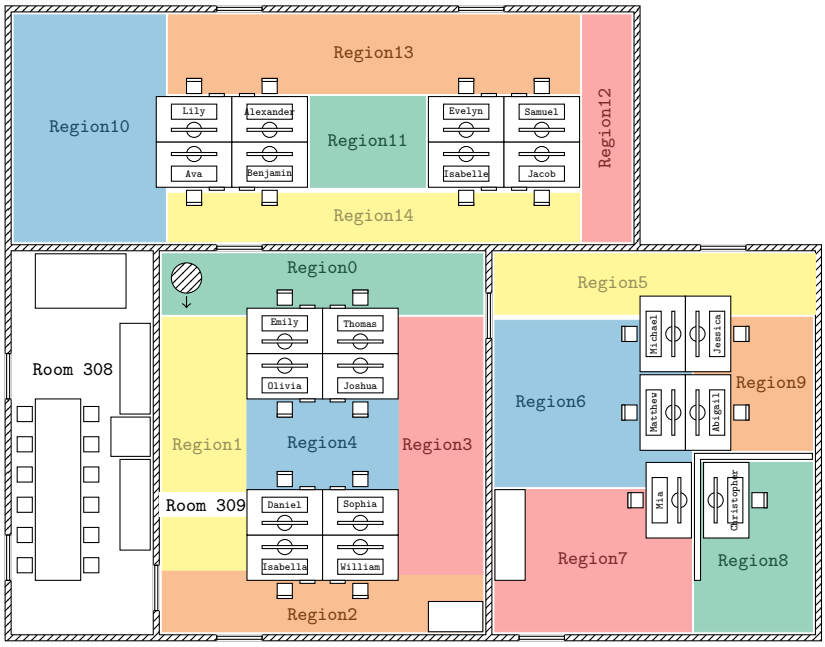

Fig. 3. The test environment

The study room is divided into 4 rooms, from left to right: the common room (not considered for the current implementation), room 309, room 310 and at the top room 314. There are workstations and cabinets in each of the rooms. They were divided into hard-coded regions based on the location of the desks, trying to find the right balance between the whole room and excessive granularity.

Regions for the moment are only areas in the environment while they are terms for the agent. Obviously, the relationships between regions considered by the agent must respect what is present in the environment and vice versa, and the agent must always have an understanding of the region it is in.

2.2 Agent: Design and Implementation

The agent is based on the BDI paradigm and knows the entire map through RCC. We want to avoid in any way that the agent has Cartesian cognition of the space it is in.

The agent aims to remove all the dirt in the room and has two plans for doing so:

1. *clean*: removes the dirt;
2. *reach*: a plan that tells the body which region it wants to reach among those adjacent to the one it is in.

The agent must know the logical map of the room and updates the region it is in as it goes, always keeping the term as ground as possible, the smallest region occupied by the agent. Ultimately, the agent must be able to reason about the paths to be taken to reach its goal. To do this requires an additional plan to be interrogated to find a sequence of noncyclic adiancent regions from the current region to the desired region. The agent must then have a way to connect to his or her body to send requests and receive information.

The implementation was done using JaCaMo. As a first step, the agent creates an artifact that establishes the connection with the physical body. The artifact then turns out to be owned by the agent, making this direct connection between agent and body without further intermediaries. The agent then executes the `find_and_clean` plan, which aims to remove all the dirt in the room.

The plan has three variants that are recursively referred to. The first has as preconditions that there is still at least one dirt to be cleaned and that that dirt is in the same region as the agent. In this case the agent must ask the body to reach the dirt and once it reaches it, clean it. When it receives confirmation of the cleaning it removes that dirt from its beliefs and calls `find_and_clean`.

```
+!go_to_clean
    :   ntpp(Dirt, Region) & my_region(Region) & dirt(Dirt)
    <-  gain(Dirt);
        .wait({+gained});
        clean(Dirt);
        .wait({+cleaned});
        -ntpp(Dirt, Region);
        !go_to_clean.
```

The second, on the other hand, only predicts that there is still dirt. If the agent uses this plan, it is because the dirt is not in the same region and therefore must move to the region where the dirt is contained. To do this, the agent has another plan `move_to` that allows it to reach the region where the dirt is.

```
+!go_to_clean
    :   ntpp(Dirt, Region) & dirt(Dirt)
    <-  !move_to(Region);
        !go_to_clean.
```

The `move_to` plan also has variations. In particular, it must take into account the room in which it is located; if in fact the region sought is in another room, the agent must look for a door that will take him into it, reach it, and then once inside it find the path to the region in which the dirt is located or another door if necessary. If, on the other hand, he is in the same room, he must find and follow the sequence of regions that lead him to the desired one. The `move_to` plan, in addition to finding the path is also concerned with following it, that is, waiting until a message has arrived for each region from the body indicating that it has been reached before moving on to consider the next one with the `follow_path` plan.

```
+!move_to(region)
    :   my_region(CurrentRegion) &
        not sameroom(CurrentRegion, Region) &
        ntpp(Region, Room) & ec(Door, Room)
    <-  ?find_path(CurrentRegion, Door, Path);
        .reverse(Path, ReversePath);
        !follow_path(ReversePath);
        !move_to(Region).

+!move_to(Region)
    :   my_region(CurrentRegion)
    <-  ?find_path(CurrentRegion, Region, Path);
        .reverse(Path, ReversePath);
        !follow_path(ReversePath).
```

For the time being, no path search optimization logic is implemented but simply the agent searches for one that is walkable.

Finally, the agent has a `clean` plan that allows it to remove dirt by communicating to its body to perform the action. This plan is called when the agent is on dirt and is the one that eliminates it. It has no preconditions, because indeed a robot vacuum cleaner can vacuum even where it is already clean, it is the `find_and_clean` plan that calls it at the right time.

Finally, communication with Godot is implemented within the Java artifact that implements a Websocket client and can send two types of messages:

- `{"type": "gain", "target": Target}`: sent to reach a Target region;
- `{"type": "clean", "target": Target}`: sent to clean the Target dirt.

2.3 Environment: Design and Implementation

The environment must represent the map presented in Figure2.1 with the due physical characteristics of the objects and walls and must handle agent displacement and dirt removal. The regions must be inserted into the scene in a physical way: the robot must not reason about them, it must find path and realize it has reached the region, so they must be something the robot can virtually interact with. Note that regions also need to be inserted at doors that make it possible to pass from one room to another.

The environment was implemented with Godot. Godot is an open-source game development software that allows to place objects within a 3D virtual environment and attach scripts to some objects. The environment was built as visible in Figure2.3 respecting the features shown in Figure2.1. Regions have been implemented in this version as `Marker3D`s and thus as individual points near which the agent must pass. The robot is implemented as `CharacterBody3D` and has attached a script that implements agent connection, movement, and dirt removal.

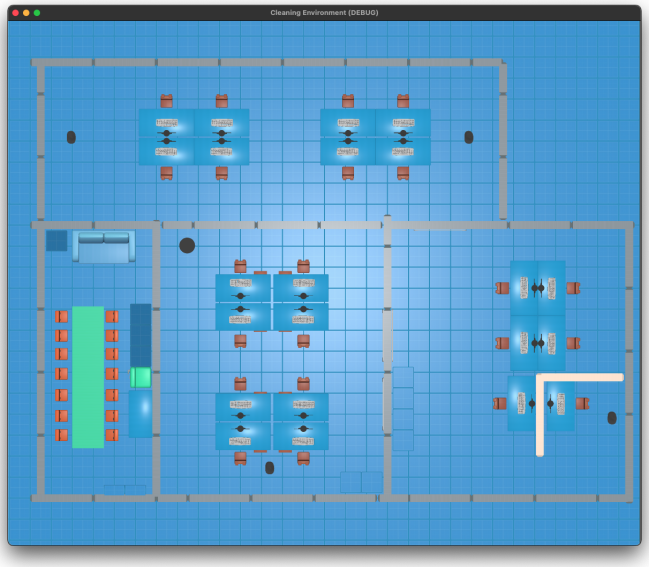

Fig. 4. The Godot environment

The `NavigationRegion3D` node was used to implement path finding. This type of node creates a map of walkable surfaces from the nodes it has as children. In our case the entire environment except the vacuum cleaner itself is instantiated as a child of this node, making everything walkable. The robot is then implemented using the `CharacterBody3D` node as its main with a `NavigationAgent3D`

as its child that implements the robot's walk on the map given by the NavigationRegion3D. The algorithm that is used by the `NavigationAgent3D` to find the best path is a version of A* on the continuum with steps of unfixed length.

The connection is implemented as a WebSocket Server directly within the script so again the robot communicates directly with its agent. This creates a 1-to-1 agent-robot correlation that may be useless for the moment but is game changing as we move from a single agent to having multiple agents and multiple robots in the same environment.

When the agent sends a message to the robot of type `{"type": "gain", "target": "regionX"}` the received region is entered as the target of NavigatAgent3D. It then calculates the shortestPath to reach it and follows it step by step. The robot's degrees of rotation are not constrained in any way, nor is the length of the space it travels at each step. When it has reached the goal it sends a message `{"type": "inform", "code": "gained"}` to the agent informing it that it is ready to perform the next task, be it moving to another region or cleaning up. If the agent requests cleaning then the node depicting dirt from the robot is destroyed and a message `{"type": "inform", "code": "cleaned"}` is sent to the agent awaiting new instructions.

3 Results

4 Conclusions and Future Work

References

1. Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach (4th Edition)*. Pearson, 2020.
2. Godot engine. `https://godotengine.org/`. Accessed: July 10, 2024.
3. Unity real-time development platform. `https://unity.com/`. Accessed: July 10, 2024.
4. Unreal engine. `https://www.unrealengine.com/`. Accessed: July 10, 2024.
5. Karthik Karur, Nitin Sharma, Chinmay Dharmatti, and Joshua E. Siegel. A survey of path planning algorithms for mobile robots. *Vehicles*, 2021.
6. Andrea Gatti Angelo Ferrando and Viviana Mascardi. Geometric and spatial reasoning in bdi agents: a survey. In *CEUR Workshop Proceedings*, 2024.
7. David A. Randell, Zhan Cui, and Anthony G. Cohn. A spatial logic based on regions and connection. In Bernhard Nebel, Charles Rich, and William R. Swartout, editors, *Proceedings of the 3rd International Conference on Principles of Knowledge Representation and Reasoning (KR'92). Cambridge, MA, USA, October 25-29, 1992*, pages 165–176. Morgan Kaufmann, 1992.