Multi-Agent Control of Industrial Robot Vacuum Cleaners

Joe Collenette[1] and Brian Logan[2]

[1] Department of Computer Science, University of Liverpool, UK
j.m.collenette@liverpool.ac.uk
[2] School of Computer Science, University of Nottingham, UK
brian.logan@nottingham.ac.uk

Abstract. In this paper, we describe a prototype multi-agent-based system for cleaning food production facilities developed as part of the RoboClean project. The prototype system is based on domestic robot vacuum cleaners equipped with IR allergen sensors and Amazon echo dot speech interfaces. The robots are controlled by a multi-agent system implemented in Jason, which handles (ad hoc) task allocation and robot coordination. We briefly describe the architecture of the RoboClean system, how coordination is achieved using the contract net protocol, and the implementation of the current prototype.

1 Introduction

Hygiene and the avoidance of cross contamination, e.g., by allergens, is very important in food manufacturing. Production and/or specialist cleaning staff typically spend a significant amount of time cleaning food production facilities, following industry standards such as those specified by the British Retail Consortium [3]. This has a significant, and increasing impact on employee productivity and costs. The drive by manufacturers to provide more variety and alternative formulations (e.g., gluten free foods) increases the amount of time that must be spent cleaning, and the potential for accidents. For example, data from the UK Food Standards Agency shows that the number of food safety events relating to allergens approximately doubled between 2014/15 and 2017/18 [5].

One possible way of reducing the amount of time staff spend on cleaning is to use robots to automate part of the cleaning task. Cleaning robots, e.g., vacuum cleaners, are becoming increasingly common in domestic environments and are starting to appear in industrial settings. However, such robots are typically designed to operate in isolation rather than to assist human cleaners, and provide limited support for the integration of ad hoc cleaning tasks into a cleaning schedule. In addition, operation typically involves either physical contact with the robot (to push a button) or a touchscreen (e.g., app-based interfaces) which may be undesirable for reasons of hygiene. Finally, such systems are not designed for a food production environment, where the type of material being removed may be significant, e.g., an allergen.

In this paper, we describe a prototype multi-agent-based system for cleaning food production facilities developed as part of the RoboClean project. The aim of the RoboClean project to investigate the potential of human-robot and multi-robot teams equipped with speech interfaces and allergen sensing capabilities for the cleaning of food production facilities. The RoboClean prototype system is based on domestic robot vacuum cleaners equipped with infrared (IR) allergen sensors and Amazon echo dot speech interfaces. The robots are controlled by a multi-agent system implemented in Jason [1], which handles (ad hoc) task allocation and robot coordination. We briefly describe the architecture of the RoboClean system, how coordination is achieved using the contract net protocol, and the implementation of the current prototype.

The remainder of the paper is organised as follows. In Section 2 we present the architecture of the RoboClean system. In Section 3 we briefly describe the implementation of current prototype and illustrate the operation of the task allocation system with an example. We conclude in Section 4 with some directions for future work.

2 The RoboClean Architecture

The focus of the RoboClean project is on human-robot interaction, flexible teamwork, and allergen monitoring rather than the practical issues related to cleaning in a food production facility, which may involve cleaning large amounts of semi-liquid material. For simplicity, we assume the materials to be cleaned are dry powders, e.g., flours, spice blends, tea, coffee etc., possibly containing allergens, such as gluten or peanut flour, and the prototype system is based on domestic robot vacuum cleaners (Neato Botvac D7 Connected[3]) augmented with an near-infrared allergen sensor (NIRONE S2.0) and a basic speech interface (Amazon Echo Dot). Similarly, for ease of implementation, the agents run on a standard PC and communicate with the robots via an API, rather than on embedded processors on the robot vacuum cleaners. While simple and cheap, the use of off-the-shelf domestic vacuum cleaners introduces a number of challenges as detailed below. The RoboClean architecture is shown in Figure 1.

2.1 Food Production Facility

A food production facility is assumed to have a variable number of cleaning robots, possibly of different types. Depending on the cleaning cycle and process, different types of robots may be used at different times (in the prototype system, all robots are of the same type, but the architecture does not rely on this), and robots may have to be taken out of service for maintenance, e.g., emptying the dust container. The production facility also defines a set of 'cleaning zones' specifying areas to be cleaned, for example, the area in front of a particular

[3] `https://www.neatorobotics.com/gb/robot-vacuum/botvac-connected-series/botvac-d7-connected/`

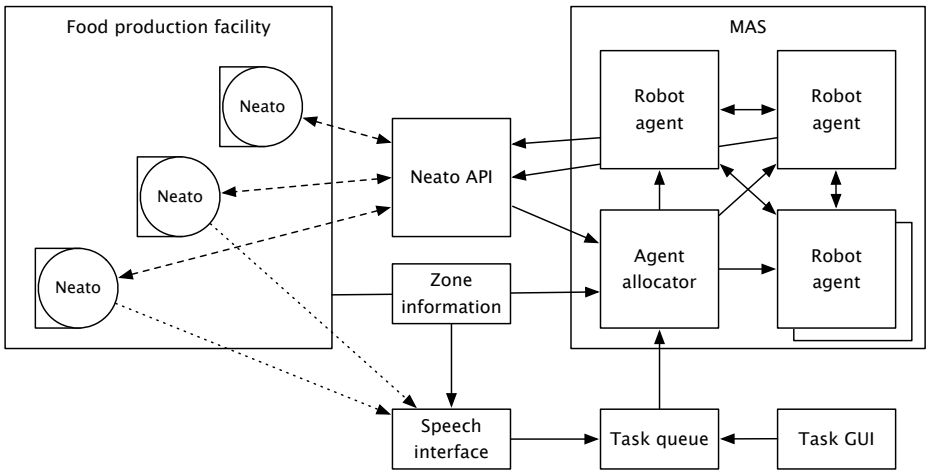

Fig. 1. The RoboClean architecture

machine or alongside a production line. This zone information is used both by the speech interface to identify the location to be cleaned, e.g., "clean next to the coffee roaster", and by the MAS during task allocation. A zone is a three-tuple and is defined as:

ID the identifier of the zone;
X, Y the coordinates of the top left hand corner of the zone; and
X1, Y1 the coordinates of the bottom right hand corner of the zone.

Zones can overlap and be nested one inside another. (The zone definition is based on that used by the Neato API.)

Regular cleaning of the facility is specified as a set of cleaning tasks. A task is four-tuple defined as follows:

ID the identifier of the the task;
Zone the name of the zone to be cleaned;
Deadline the time by which the task should be completed; and
Priority the importance of the task is relative to other tasks (smaller numbers indicate a higher priority).

Tasks are specified using a simple GUI and added to a task queue.

The Neato Botvac D7 robots are internet connected robotic vacuum cleaners which can be controlled using mobile devices, e.g., smartphones, and other smart home devices such as Amazon Alexa and Google Home. The robots are battery powered, and recharge at a base station. Each robot is approximately 30cm $\times$ 30cm and weighs 3.5 kilograms. It has a front bumper that detects collisions and a top mounted laser used for both mapping and navigation. In the RoboClean system, each robot also has a speech interface which can be used to query and control the robot and the robot team, and an IR sensor. (In the interests of

simplicity, the IR sensors are not shown; they are currently not integrated with the API and communicate indirectly with the MAS via a bluetooth connection.[4]) The speech interface can be used to give the robot (or a robot team) ad hoc cleaning tasks, which are added to the task queue.

2.2 Multi-Agent System

The multi-agent system consists of an Allocator Agent and a variable number of Robot Agents. The allocator agent has two main roles: it monitors the Neato API (described below) and, when a cleaning robot comes online, it allocates a robot agent to control the robot (creating the robot agent if necessary); it also monitors the task queue and dispatches new tasks to robot agents. Each robot agent is responsible for monitoring and controlling a cleaning robot. The agent periodically polls its allocated robot to check its connection, battery and cleaning status, and issues commands as necessary to perform cleaning tasks. Robot agents are also responsible for task allocation and task achievement.

Tasks are allocated using a version of the contract net protocol [9]. In the contract net protocol, each task has a manager who announces the task to other agents and requests bids, and then allocates an agent to perform the task. All agents able to perform the task (including the task manager itself) send a bid to the manager containing the agent's estimate of how long it would take it to perform the task, taking into account its current location relative to the task, charge level, dust container capacity and the tasks to which it is already committed. When all eligible agents have returned bids, the manager allocates the task to the agent that can perform the task in the least time. That agent then adds the task to its task list. The task will either be performed immediately (if the robot controlled by allocated agent is currently idle) or scheduled for future execution (e.g., after currently executing task(s) with earlier deadlines). When the task has been completed, the agent allocated the task notifies the task manager, which in turn notifies the allocator agent to update the interface. In the RoboClean architecture, when a task is added to the system task queue, e.g., an ad hoc task requested by a member of the production staff via the speech interface, the allocator agent randomly selects a robot agent to act as task manager. The selected robot agent remains responsible for the task until notified of its completion. If the allocated agent is unable to complete the task, e.g., because the robot it is controlling goes offline, it notifies the task managers of all its allocated tasks so that the tasks can be re-advertised and allocated to other robot agents. Once the task managers have been notified, the robot agent then notifies the allocator agent that the robot that it is controlling is offline, and its status is changed to 'unallocated'.

2.3 Neato API

The Neato API forms the interface between the MAS and the cleaning robots. The Neato robots expose only a high-level webservice/IoT API, primarily in-

[4] For details of the sensor, see [8].

tended for developing apps (e.g., for mobile devices). The API allows basic information about the robot to be queried, e.g., whether it is cleaning or charging at the base station, battery level etc., and provides some high-level commands, e.g., start/stop, clean zone X, etc. However, using the API, it is not possible to obtain sensor or position information from the robot, or execute low-level actions, e.g., moving to an arbitrary location. After cleaning a zone, the robot will return to the base station before starting to clean the next zone. These restrictions impact the coordination and control possible in the current prototype but not the overall architecture which is capable of finer coordination with more controllable robots.

3 RoboClean Prototype

To facilitate development and testing of the agent coordination, in the current prototype implementation of the RoboClean architecture, the Neato robots and Neato API are realised using a simulator. The simulator models the dynamics of the Neato robot vacuum cleaners, and provides the same query and control functionality as the Neato API. Similarly, the speech interface is modelled using a process that randomly generates ad hoc tasks which are added to the task queue. The architecture of the prototype is shown in Figure 2, and each of the components is discussed in detail below.

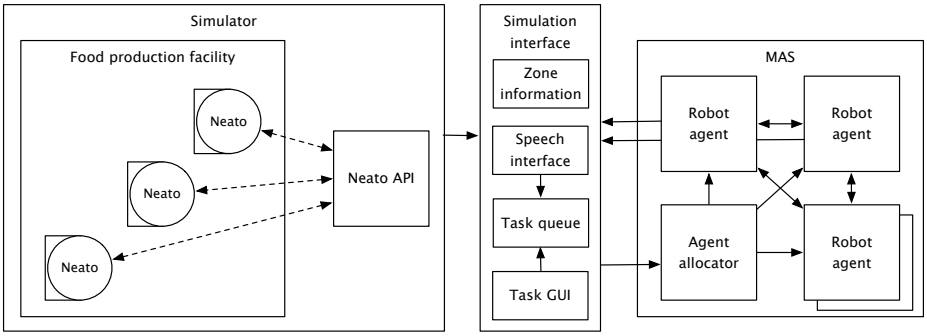

Fig. 2. The RoboClean prototype

3.1 Simulator

The environment is represented by a grid of cells, where each cell is 33cm (i.e., the size of a robot). The layout of the environment and the number of robots to be simulated is specified in a text file. The first line of the file contains the key simulation parameters, and the remaining lines specify the contents of each cell. The simulation parameters are:

Dimensions the size of the environment in x and y (in cells);
Robot Count the number of robots; and
Simulation Speed the time between each simulation step in milliseconds.

The contents of each cell are specified using a simple textual encoding of size $x \times y$ where:

E represents an empty space;
O represents an obstacle (a space that a robot cannot occupy); and
B represents base station.

The number of base stations must be the same as the number of robots in the simulation and each robot is initially located at a base station to which it returns to recharge. In the simulator interface, robots are shown as blue circles, empty cells in light grey, obstacles in dark grey and base stations in yellow. During the simulation, the simulator randomly generates 'dirt' in empty cells, indicated by green cells. For example, the environment specified below is depicted in Figure 3.

```
10 10 3 100
BEEEEEEEEB
EEOOEEEEEE
EOEEEEEEEE
EEEEEOEEEE
EEEEEEEEEE
EEEEEEEEEE
EEEEEEOEEE
EOEEEOEEEE
EOEEEEEEEE
EEEEEEEOEB
```

A more realistic environment, based on a food production facility is shown in Figure 4: the dark grey curved lines represent the conveyor belts and processing stations where food is prepared.

As explained in Section 3.3 below, the task allocation algorithm assumes that the travel and cleaning times for a given zone are available. In the prototype, this information is computed by the simulator. When Neato robot receives a command from the Neato API to clean a zone, it uses its map of the environment to compute the shortest route to the top left corner of the zone. It then turns on the vacuum and begins cleaning the zone in a 'reverse S' pattern. That is, starting at the top left, it cleans the top row of cells left to right before moving down a row and cleaning the second row from the right to left and so on, continuing until the entire zone has been cleaned. The robots move more slowly when cleaning than when travelling between the base station and zone. The simulator mimics the behaviour of the physical robots, allowing relative travel and cleaning times to be calculated. The travel time is proportional to the minimum number of cells that must be traversed by the robot to reach its base station plus the number of cells from the base station to the first cell of the zone to be cleaned. The time required to clean a zone is assumed to be the size of zone times 10, as the Neato is approximately 10 times slower when the vacuum is engaged.

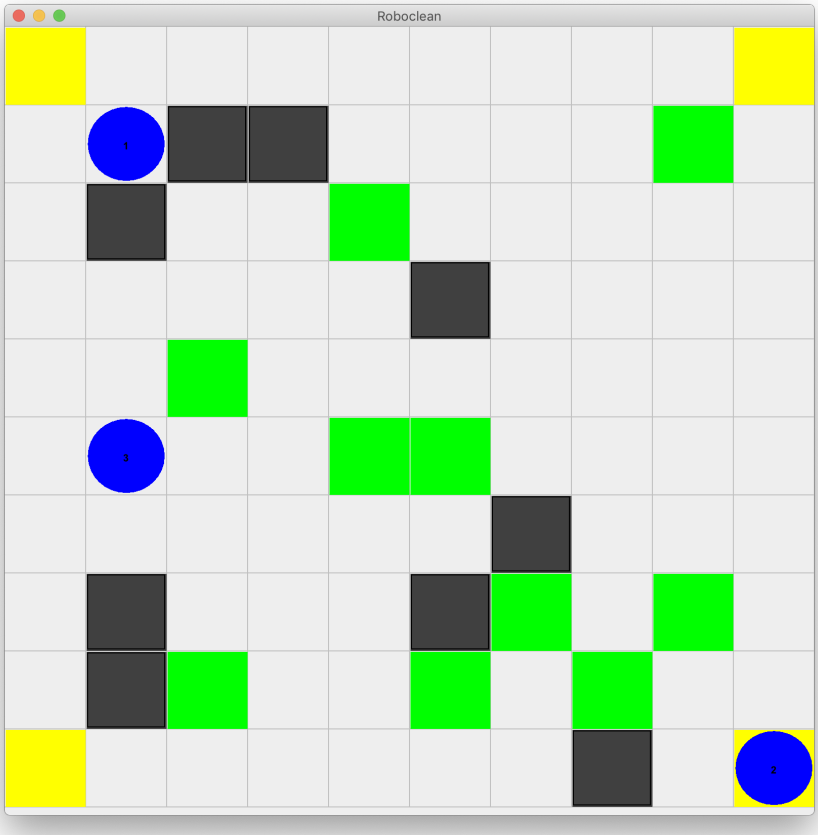

Fig. 3. Example of simulation environment

3.2 Simulator Interface

The simulator interface manages the connection between the agents and the simulated environment. It also provides a simple interface to the task queue and the zone information defining the areas of the simulated environment that may be cleaned. The initial list of scheduled tasks can be updated at run time, simulating the effect of ad hoc tasks from the speech interface. The simulator interface GUI is shown in Figure 5. The panel on the left contains the zone definitions, and the panel on the right the current task queue. As explained above, each task is specified by a zone to be cleaned, a deadline and a priority.

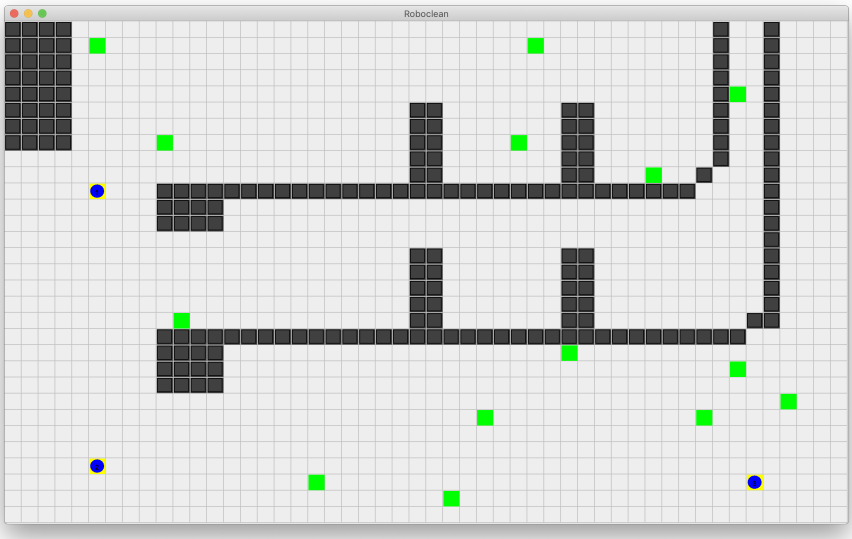

Fig. 4. An example food production facility, with robots shown at their base stations.

3.3 Multi-Agent System

This allocator agent and robot agents are implemented in Jason [1]. The contract net implementation makes use of the Jason contract net library. When an agent i receives an announcement of a task j, they compute a bid, i.e., the time it will take to complete the task, using the the equation below:

$$Bid_i^j = \begin{cases} \infty & \text{if busy} \\ (MoveTime_i^j + CleanTime_i^j) * AllocatedTasks_i & \text{otherwise} \end{cases}$$

where $MoveTime$ is the amount of time it would take to move to the zone to be cleaned and $CleanTime$ is the amount of time it would take to clean the zone. The total time required for zone j is weighted by the number of tasks already allocated to agent i; this approximates the time required to return to the base station after cleaning the zone, recharging time etc.. While simplistic, this approach maximises the number of robots actively working on tasks (which users studies suggested was preferred by production staff, even if the resulting allocation is not optimal). The order the tasks are allocated is firstly by priority, then secondly by deadline. To ensure that the agents are portable between different robots, environments, and tasks we assume that that times are available from the robot or the robot API (in the prototype, the simulation interface).

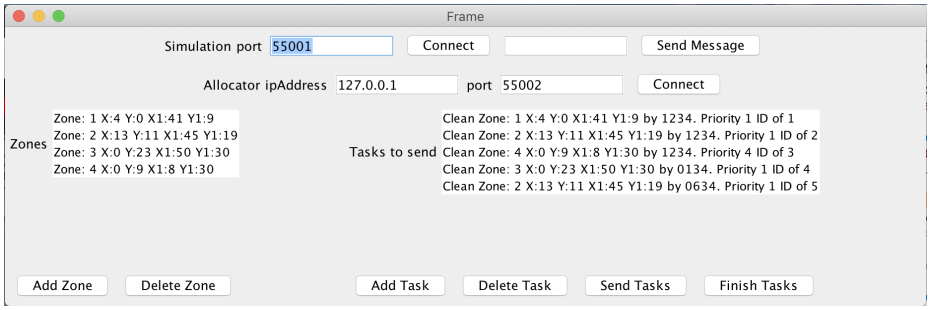

Fig. 5. The Intermediate Manager program with Zones and Tasks that will be used in our examples.

3.4 Example

As an example, we will show how the tasks are allocated when the task list is sent in two different scenarios. The first scenario is when the system is in the initial phase when no tasks have been allocated. The second scenario is when some, but not all, tasks that have been allocated have been completed. Before we can allocate the tasks we need to know what order they will be processed in. One possible ordering, where the first to be processed is at the top, is:

Task 4 Clean Zone 3, by 01:00 with a priority 1
Task 5 Clean Zone 2, by 06:00 with a priority 1
Task 1 Clean Zone 1, by 12:00 with a priority 1
Task 2 Clean Zone 2, by 12:00 with a priority 1
Task 3 Clean Zone 4, by 12:00 with a priority 4

The order in which tasks 1 and 2 are processed may be swapped, since they both have the same deadline and the same priority. The order in which they are processed depends on which task was received first by the agents. Below we assume that the tasks are received in the order given by the list.

Initial Allocation We start with an example of task allocation in the initial start-up environment, where there are no previous tasks. The first task to allocate will be Task 4, as this task has the highest priority and the earliest deadline, making it the most important to allocate.

The first set of bids will come in from the three agents for the first task to be allocated (Task 4).

$$\begin{aligned} bid_{A1}^{T4} &= 0, \\ bid_{A2}^{T4} &= 0, \\ bid_{A3}^{T4} &= 0, \end{aligned} \tag{1}$$

All the bids are 0, as no task as been allocated to any of the agents. The bids being weighted against the number of previous tasks forces all the agents to place the best bid. The task can then be assigned to any of the agents, for our example we will assume that it has been assigned to Agent 1.

Task 5 is the next task to be allocated. Agent 1 will place a bid which is non zero, as this agent has been allocated a task. Agents 2 and 3 both place the best bids. For out example we will assume that Agent 2 has been allocated Task 5. Similarly for the next task, Task 1, Agent 3 will place the best bid, as this agent is the only one not allocated a task.

At this point the allocated tasks are:

Agent 1 Task 4
Agent 2 Task 5
Agent 3 Task 1

Task 2 is the next task to be allocated. The bids from the agents will now not be 0, as they all have at least one task allocated to them. For Task 2 the bids from the 3 agents will be shown in Equation 2, which is Equation 3.3 with variables solved for our example.

$$\begin{aligned} bid_{A1}^{T2} &= (14 + 2560) * 1, \\ bid_{A2}^{T2} &= (24 + 2560) * 1, \\ bid_{A3}^{T2} &= (58 + 2560) * 1, \end{aligned} \tag{2}$$

Agent 1 has produced the lowest bid as it is closest to the starting point of zone 2. Task 2 will be allocated to Agent 1. The final task to be allocated is Task 3. The bids for Task 3 will be:

$$\begin{aligned} bid_{A1}^{T3} &= (7 + 2880) * 2, \\ bid_{A2}^{T3} &= (24 + 2880) * 1, \\ bid_{A3}^{T3} &= (63 + 2880) * 1, \end{aligned} \tag{3}$$

While normally Agent 1 would be assigned Task 3, as it is the closest, the agent also has the most tasks allocated. The next closest is Agent 2, which has also put in the best bid for the task. The final allocation of tasks is therefore:

Agent 1 Task 4, Task 2
Agent 2 Task 5, Task 3
Agent 3 Task 1

Additional Task Allocation The aim of the second example is to show how the agents and the contract net cope when presented with an additional list of cleaning tasks when they are currently working on a previously allocated set of tasks.

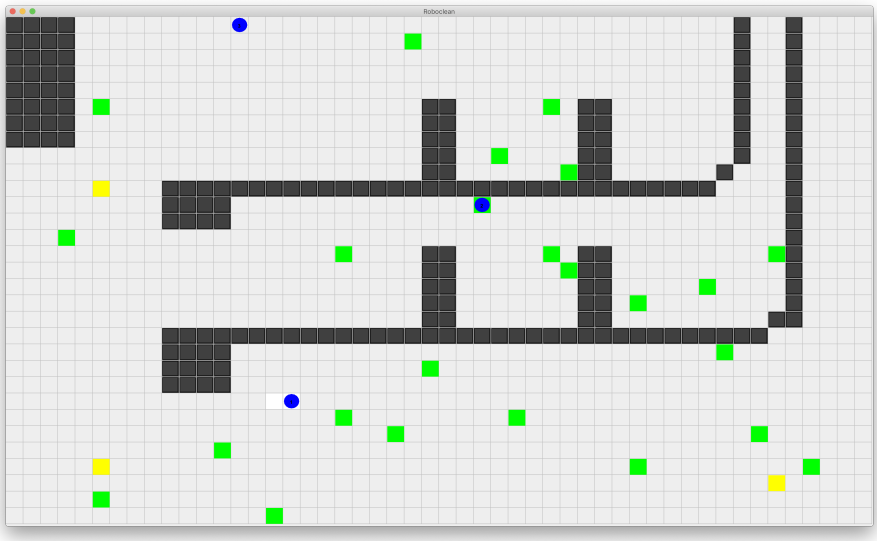

Fig. 6. Factory simulation environment, where the simulated agents are currently moving around the environment working on cleaning tasks.

At this point in our example the agents will have sent their robot representative off to achieve one of the cleaning tasks that they have been allocated. The human operator in the food factory has noticed that there has been a spillage on the factory floor and have requested that another two tasks need to be allocated:

Task 1 Clean Zone 4, by 12:00 with a priority 1
Task 2 Clean Zone 3, by 12:00 with a priority 1

Both the tasks requested have the same deadline, they also have the same priority. Therefore the task that will be allocated first will the task that was received by the manager first. In this example we will assume that Task 1 is the task that will be allocated first.

We will assume that the state of the environment is the same as presented in Figure 6. The main feature to note is that no simulated robot is currently at the base station and all the agents are either working on one of their tasks or are moving back towards their base station. The agents are assumed to have the current tasks that still need to be completed:

Agent 1 Task 2
Agent 2 Task 3
Agent 3 All tasks completed

Each agent has completed a single task from the tasks that were allocated in the first example. Agents 1 and 2's robots are currently working on the next task, while Agent 3's robot heads back towards the base station.

The allocation for Task 1 is simple to calculate when the manager requests all the bids. Agent 3 will be the only agent to return the best bid of 0, since it is the only agent that has no tasks currently assigned to it. Task 1 will be assigned to Agent 3.

Agent 1 Task 2
Agent 2 Task 3
Agent 3 Task 1

Agent 3 now has a task to complete, but it is unable to send the robot to start the task until the robot has returned to the base station. The manager will move on to assigning Task 2. The bids for this task will be:

$$\begin{aligned} bid_{A1}^{T2} &= (34 + 2560) * 1, \\ bid_{A2}^{T2} &= (48 + 2560) * 1, \\ bid_{A3}^{T2} &= (83 + 2560) * 1, \end{aligned} \tag{4}$$

At this point it is worth noting that the simulation calculates the time needed to get back to the base station as well as the time needed to get to the zone to clean. In simulation, Task 2 would be assigned to Agent 2 as the base station is the closest to Zone 3.[5]

Therefore at this point the current tasks to complete for the agents in the simulation are:

Agent 1 Task 4, Task 2
Agent 2 Task 5
Agent 3 Task 1

All the tasks have been allocated. Given the speed at which the tasks can be allocated, we can assume that the state of the world has not changed between the start of allocation and the end of allocation. When the robot Agent 3 represents returns to the base station, the agent will be able to send the newly assigned task to its robot.

4 Discussion

We have presented a prototype allergen aware factory cleaning system, which is part of a larger RoboClean project that aims to facilitate effective and efficient cleaning through multi-agent and human-robot interactions. Our prototype system allows a queue of cleaning tasks to be distributed among a number of robots using the contract net protocol [9]. The contract net protocol was chosen due

[5] When the prototype is implemented on real Neato robots, it will only be able calculate the move time based on the time it would take to move from the base station, as the Neato does not reveal its location through the API.

to its simplicity and the relatively small amount of information and communication required. There are a number of other task assignment protocols that extend contract nets, such as Alliance [7] and M+ [2], and scheduling approaches that focus on either minimising the number of late jobs [4] or taking, e.g., the battery life of the robot into consideration [6]. However, given the limited information available via the Neato API, we believe the contract net is a reasonable approach.

In future work, we plan to interface the MAS to the Neato API and hence to control the physical robots. This will provide a platform for user studies investigating human-robot interaction to be explored, e.g., in which situations does a human view themselves as interacting with a single robot and in which situations do they see themselves interacting with the team of robots through the robot being addressed. Another area of future work would be investigate alternative robot platforms which allow finer-grained control. This would allow a better allocation of tasks, e.g., in terms of minimising cleaning time, or ensure all tasks are completed before the deadline.

References

1. Bordini, R.H., Hübner, J.F., Wooldridge, M.: Programming multi-agent systems in AgentSpeak using Jason, vol. 8. John Wiley & Sons (2007)
2. Botelho, S.C., Alami, R.: M+: a scheme for multi-robot cooperation through negotiated task allocation and achievement. In: Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No. 99CH36288C). vol. 2, pp. 1234–1239. IEEE (1999)
3. British Research Consortium: Global standard food safety - issue 7 (2015)
4. Brucker, P.: Scheduling algorithms. Springer (2007)
5. Food Standards Agency: Annual report of food incidents 2016/17 (2018)
6. Luo, L., Chakraborty, N., Sycara, K.: Distributed algorithms for multirobot task assignment with task deadline constraints. IEEE Transactions on Automation Science and Engineering **12**(3), 876–888 (2015)
7. Parker, L.E.: Alliance: An architecture for fault tolerant multirobot cooperation. IEEE transactions on robotics and automation **14**(2), 220–240 (1998)
8. Rady, A., Fischer, J., Reeves, S., Logan, B., Watson, N.J.: The effect of light intensity, sensor height, and spectral pre-processing methods when using NIR spectroscopy to identify different allergen-containing powdered foods. Sensors **20**(1) (2020)
9. Smith, R.G.: The contract net protocol: High-level communication and control in a distributed problem solver. IEEE Transactions on computers (12), 1104–1113 (1980)