An initial solution to MAPC 2019

The ROS hybrid planner based approach

Dayyan Smith · Mayank P. Yadav · Vinit Jain · Taha Jirjees Ahmed

21.07.2019

Abstract Our team's solution to the MAPC 2019 makes use of ROS and RHBP to have multiple agents work on a strategy to submit as many tasks as possible. The agents communicate with each other and use their perception to merge their local maps with those of other agents to get a global view of the map. The hybrid planner helps agents to take the best suited action to complete this strategy. Agents are able to do path planning by making use the A-star algorithm. Our task submission strategy makes the assumption that all tasks have only a one block submission requirement.

Keywords Multi-Agent Programming · Artificial Intelligence · RHBP · ROS

1 Introduction

Intelligent agents perceive their environment and adapt accordingly to achieve the goal. Multi-agent systems (MAS) comprise of several intelligent agents to finish tasks which is not possible for a single intelligent agent. Nowadays, Multi-agent systems became preponderant among computer simulation tools [9]. Solving a problem in multi-agent system varies according to the simulation and entities which leads to complexity of a system given the number of agents, behaviour and their interaction. For instance [6] focuses on use of reinforcement learning and evolutionary computation for team learning. [4] introduced a new ROS Hybrid Behaviour(RHBP) for task level decision making and behaviour planning for each agent. [8] also showed a different approach where the use of game theory, CLRI theory and n-level learning agents is used in MAS to attain the task.

In this paper, we present the solution overview of the Multi-agent Programming Contest 2019 presented by the team *Storm-troopers*. In the scenario, Agents

Dayyan Smith · Mayank Yadav · Vinit Jain · Taha Jirjees Ahmed
DAI Labor, Technische Universität Berlin
Ernst-Reuter-Platz 7
10587 Berlin / Germany
Tel.: +49 30 314 74000
Fax: +49 30 314 74003

Assemble, agents have to collaborate together to create complex structures and submit them to goal points to attain maximum utility. For implementation, we used Robotic-Operating System(ROS) and RHBP [4] for agent behaviour and decision making. The framework was designed in DAI-Labor at Technische Universität Berlin through a course of several theses.

2 System Analysis and Design

2.1 ROS

ROS (Robot Operating System) was developed as a framework to facilitate rapid prototyping when it came to writing software for robots. The researchers in [7] mention how it was difficult to manage the wide array of different challenges researchers faced when trying to develop code for robots — ranging from the sheer volume of code required for each module as well as the non-heterogeneity of hardware components leading to lack of code re-usability.

ROS tries to address these issues as well as hold true to the design principles that [7] want their proposed framework to offer. [7] then goes on to explain the essentials in an implementation of ROS as:

- **Nodes**: akin to ‘software module’, are computation processes.
- **Messages**: a strictly typed data structure that allow communication between nodes.
- **Topics**: Nodes can publish messages topics. Other nodes interested in that particular type of data can then subscribe to it.
- **Services**: akin to ‘web services’, consist of a string name and a pair of strictly typed messages: one for the request and one for the response.[7]

2.2 RHBP Framework

As robotic systems get better and move away from strict test spaces to being more and more ingrained with our daily lives, this changing paradigm also brings its own challenges for how these robotic systems navigate, decide on tasks and generally make decisions in an uncertain and changing environment. [5]

In [5] the researchers lay out a framework to facilitate this kind of robotic systems that can work together to accomplish a goal in an environment that relies more on their ability to adapt to uncertain situations rather than to rely on preconceived notions of what these systems or agents will have to do.

The architecture defined in [5] describes their implementation so that it is able to combine a goal orientated approach for the entire system while still being able to react and adapt by use of behavior networks in combination with planning. This combination is achieved through a module [5] call the **meta level symbolic planner**. This also ties into the goal orientated approach mentioned previously by ensuring that the high-level goals that the agents should be pursuing are dependant or influenced by the low-level behavioural network and information gained by each agent. Each agent have their own symbolic planner which ensures that they are able to do tasks independently in cases where they are unable to communicate with the centralized planner. [5]

In [5] the authors highlight that the centralized planning done here only provides an outline for goal pursuance but is decoupled enough that the behavioural network of an agent is able to be opportunistic and perform the best possible action it can.

The main components in the behavior network base are **behaviours** and **goals**. [5]. A detailed breakdown as per [5] would be:

- **Network of Behaviours**: Generated from *preconditions* and *effects* or *wishes*.
- **Wish**: Range of satisfaction about the current state of sensor values. These have a real value between [-1,1] where -1 shows weaker desire, 0 indicates complete satisfaction, and 1 indicates stronger desire.
- **Sensor**: Provide information and sensor measurements to the behaviour network.
- **Activators**: Activators use values taken from sensors to compute a utility score. [5]

2.3 MAPC 2019 Scenario

The scenario for the Multi Agent Programming Contest (MAPC) 2019 called "Agents Assemble" asks teams to have their agents assemble complex structures and then deliver them to a designated area for submission [1]. The goal of the contest is to enhance the research in the field of multi-agent systems and develop software models.

The environment consists of a rectangular grid where both teams can compete with their group of agents to finish tasks (aforementioned combining of structures and submission to goal area) to earn points. [2]

Each grid can consist of a limited number of **Things**, namely: **Entities**, **Blocks** or **Dispensers**.

3 Software Architecture

3.1 Task Allocation

Each agent keeps track of all tasks in the simulation. Each agent chooses a task which is not yet completed or expired.

In our current implementation the agents do not collaborate for the task allocation. For task submission, we make the assumption that tasks have just one block to submit as a requirement. By way of this assumption, each agent is able to be assigned a task that is not yet completed or expired is able to submit the task at the goal area.

3.2 Hybrid Planner

The actions an agent takes come from a hybrid planner combining a behaviour network and a symbolic planner, which enables pursuance while maintaining reactive and adaptive behaviour-based decision making capabilities [5]. In our case the goal for each agent was to submit as many tasks as possible. To achieve this

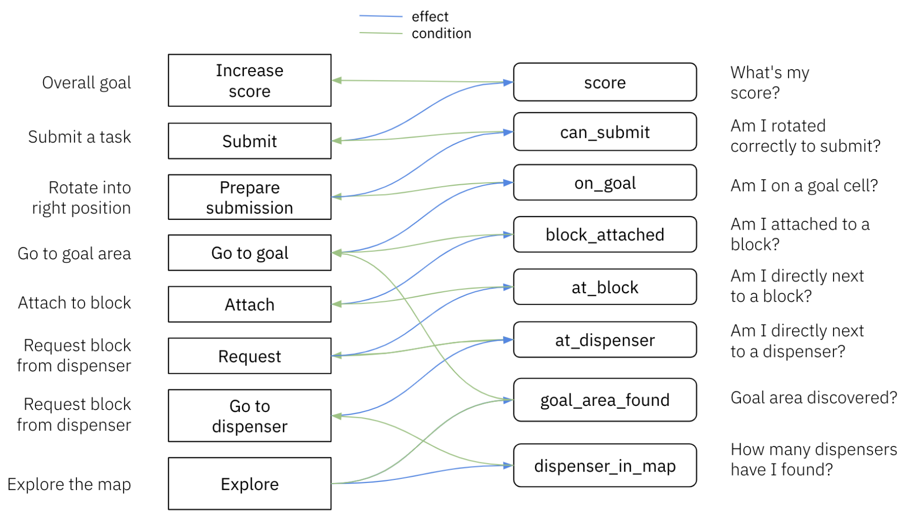

Fig. 1: Behaviours and sensors

functionality, we implemented several behaviours that are linked together by the conditions that need to be fulfilled for any given behaviour to be executed and the the expected effect they have. Underlying these conditions and effects are the sensors we implemented. These sensors encode information the agent has about the world, its task and its current state. The conditions of each behaviour are the values a sensor needs to have in order for a behaviour to be chosen. The effects of each behaviour encode the expected effect a behaviour has on the sensor values.

Figure 1 shows the general idea of how the overall goal of increasing the score can be achieved by executing a sequence of behaviours. The order of the sequence of behaviours (on the left) is determined by each behaviours conditions and expected effect on the sensors (on the right). To increase the score, an agent with the behaviours and sensors given in this figure would act like this: Initially it explores the world and it continues doing so until there is an appropriate dispenser in its map. This dispenser can appear because of the agent's own exploration, or because of map merging with another agent's map. Once a dispenser of the correct block type for the agent's current task is in the agent's map, the agent can move towards that dispenser. When the situation of an agent at the dispenser requesting a block becomes possible, and then, when the request is successful and a block is dispensed, the agent is at the block and can attach to it. At this point the agent can go to the goal, if its location is already discovered. Otherwise more exploration is done. When the goal is discovered, the agent will move there with the attached block. Once the agent is there, it will rotate until the position of the attached block matches the position of its assigned task. Then the agent can submit the block and increase its score.

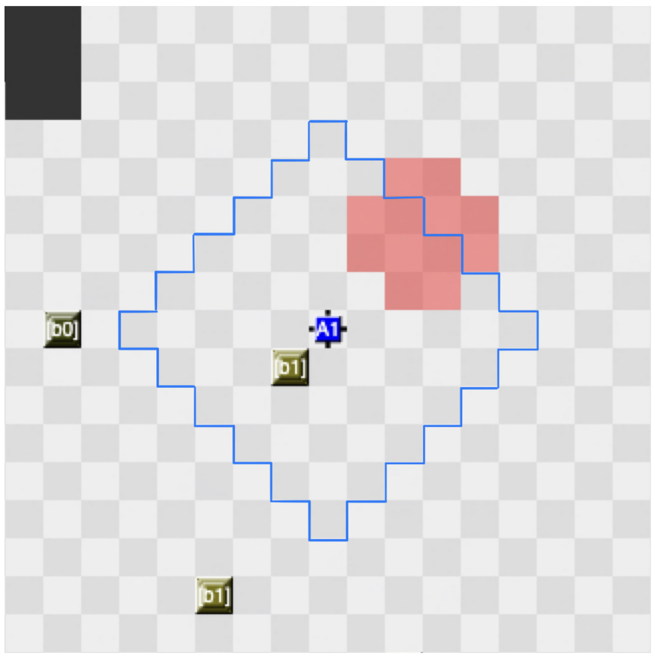

Fig. 2: Agent vision

3.3 **Mapping**

One of the key capabilities an agent needs to have to efficiently accomplish tasks is a map of the environment. In a first step ,an agent saves its perception to create a map of the area that agent has already seen. The ”local” maps of two agents can be merged when they overlap. Determining when two maps overlap is difficult. In our case, we assume the goal area to be a unique landmark so that map merging can take place as soon as two agents have discovered the goal area.

3.3.1 Agent vision and perception message

At each time step an agent receives a perception message including information about the terrain and the things in its vision. The cells in an agents vision for which there is no terrain information are empty. The vision of an agent is 5 using the Manhattan distance, resulting in a diamond shaped perception at each time step. In figure 3 agent A1 can see the dispenser of type *b1* and some of the goal cells. The other cells in the blue diamond agent A1 perceives as empty.

An agent perceives everything in this diamond, this means it can also see passed obstacles.

3.3.2 Building the local map

An agent's current perception of the world around it and the direction in which it moved to get there are used to build a map of everything the agent has seen in the world. At each time step an agent processes its perception and updates the map with the obstacles, goal cells, blocks and dispensers it perceives.

3.3.3 Merging maps

Once the maps of two agents include the goal area those maps can be merged, since the goal area is assumed to be a unique landmark. Merging the maps results in updated maps for both agents, now including areas that were previously undiscovered by one agent, but already discovered by the other. Discovering the goal area is a requirement for merging maps and as soon as an agent discovers it, the agent starts publishing its own map to a ros-topic and simultaneously subscribing to it. From this point onward at each time step an agent will publish its map at each time step and receive maps of other agents who have discovered the goal. The agent will then merge its current map with the maps it has received from other agents. Doing this at every time step ensures always having an current and accurate representation of the world.

3.4 Path Planner

One of the basic problem encountered in graph theory is the motion planning. Motion planning is one of the core research problems in robotics. Either a simulation or real-world problem, it is important to have a shortest collision free path between start and destination point. In [3], a heuristic and mathematical approach to do path planning is discussed. This algorithm uses the heuristic of distance between the current node to goal node and sum it with the distance between starting node to current node to define the motion of the robot. The algorithm is called A-star algorithm. In our simulation, we used the A-star approach to find the shortest path between current position of agent and entities such as dispenser, goal, blocks and other agents from same team. The algorithm is detailed in **Algorithm 1**.

For a graph $\boldsymbol{G(E,V)}$ with E edges and V vertices, we have *start* as starting node and *goal* as goal node to find a path. We also maintain two list:

– open : list of nodes already visited with children yet to be explored.
– close : list of nodes already visited with children explored and added to open list.

3.5 Exploration

Exploring the environment is essential for updating each agents perception and discovering entities around the map. Our implementation aimed to improve upon a random-movement exploration pattern.

To achieve is improvement our idea was to have each grid in the agent's perception be updated with a score, the score is assigned on the basis of the number of -1s in its local map as we already know that agent has a vision of diamond which means it has undiscovered areas at peripheries. This score ranges between [0,9] and the idea is that the agent will try to move towards an area where it can get the highest score, thereby maximising its chances of exploring unseen areas of the map, which for now always moves our agents northwards.

Algorithm 1: A-star path planning

```
Data: start, goal, OPEN, CLOSE
1  Put start in OPEN; $f(start)=h(start)$ initialization;
2  while OPEN is not empty do
3      read current node with lowest f(current)=h(current)+g(current);
4      if current=goal then
5          Path found;
6          break;
7      end
8      Generate node children for each current;
9      for node children in current do
10         children cost=g(current)+w(current,children);
11         if children in OPEN then
12             if g(children)is less than children cost then
13                 continue to line 27
14             end
15         end
16         else if children in CLOSE then
17             if g(children)is less than children cost then
18                 continue to line 27
19             end
20         end
21         else
22             Add children to OPEN list;
23             h(children)=w(children,goal);
24         end
25         Set g(children) to children cost;
26         Set parent of children to current;
27     end
28     Add current to CLOSE;
29 end
30 if current is not goal then
31     OPEN is empty;
32     No path;
33 end
```

3.6 Communication

Communication is an important aspect in multi-agent systems. Agents have to gather and share information with other agents to update about the environment accordingly. Setting up communication across agents makes the system centralized as every agent is aware about other agents finds the reference area. This enables other agents to make informed decisions even when the shared information is outside it's perception range. The agents share their perception about the surroundings to other agents as soon as they find the reference area. In our case, the reference area are the goal cells as they are unique in the entire grid space. This avoids conflict when the agents are communicating with each other as the agent only share the information about the environment only when the goal cells are found.

4 Implementation

The basis and most fundamental part of our implementation is mapping. We therefore first implemented local mapping for each agent and then map merging, which additionally requires a way for agents to communicate. Additionally we implemented a path planner to move around in the world while avoiding obstacles. Building on this we added a simple way of allocating tasks and combined everything with a hybrid planner to achieve our simple goal of having an agent submit as many tasks as it can.

4.1 Mapping

Each cell in the two-dimensional world can either be an empty cell, a goal cell or an obstacle. Additionally there are things that can be on free or goal cells, but not on obstacle cells. These things are agents, blocks and dispensers. Dispensers are created at the start of a simulation and cannot be moved. Agents can move themselves and attached blocks around in the world. Agents and blocks can also occupy the same cell as dispensers. We want to keep track of the terrain type of each cell and the things on the map. We chose to simultaneously create multiple representations of the world: two containing terrain information, and one each for blocks and dispensers. In the array for goals we represent goal cells with the value 1, all other cells the agent has perceived as 0 and undiscovered cells with -1. In the obstacle array obstacles are represented by 1, other cells the agent has perceived by 0 and undiscovered cells by -1. Combining these arrays gives us information on free cells. In the blocks and dispensers arrays the respective things are marked by 1, everything else by -1. Furthermore we keep track of the agent's position in the map. For this we keep track of where the agent was at the beginning of the simulation — where it spawned. And we update this spawn position accordingly while the map grows. The position of the agent relative to the spawn position is easily updated each time the agent moves.

The global position of the agent in our map (with the origin in the top left corner) can be calculated using the spawn position, which is the global position of where the agent started and the agent's position relative to the spawn position. An agent has a vision of 5, as can be seen in figure 3 and the map of an agent starts out having dimensions 11x11, so that this diamond can fit. At each following time step, if the agent moves its position in the map is updated. If its new position results in perceiving parts of the world outside of the current map, we add a row or a column to the map accordingly.

4.2 Communication

Agents start publishing their maps to a *map* topic as well subscribe to that same topic once they have discovered the goal area. Each agent publishes its map at every time step. As soon as the messages are published, every agent that has subscribed to that topic start receiving the messages. These messages are then stored to a buffer which are also processed at every time step. Since we only use one topic for the communication of maps, an agent will also "receive" its own

```
string message_id
string agent_name
string obstacles
string goals
string blocks
string dispensers
int32 height
int32 width
```

Fig. 3: Map message

messages. Therefore we include the agent name in the message to filter those messages out before adding them to the buffer. Furthermore we convert the four arrays we use to store the information of the map to comma separated strings. The height and width are used to reshape the one-dimensional representations back into two-dimensional arrays.

5 Evaluation

As mentioned in section 4.1 on how an agent perceive its environment, and when agent merges its own map with that of another agent the following happens: it calculates the gaps between the borders of the maps and the goal area to determine where the maps overlap and what are the dimensions of the new map. Then both maps are placed at the appropriate position in the new map. We use the the arrays containing information about the goal cells to determine how to merge the maps and then merge the other representations of the world in the same way. The cells that are added outside of the dimensions of the two maps which are being merged are undiscovered and therefore marked by -1 showing unexplored area.

Figure 5 represents the global map of the simulation using 5 agents. It takes 350 timestamps for the same map(40x40) without exploration strategy and 150 timestamps with the exploration strategy to move to unexplored area as referenced in section 3.5. We also tried to implement the same strategy on the smaller map of 20x20. The global map takes 60 timestamps to make the global map and share it with all other agents. It took 4 agents. The light blue portion shows the free space. Agents also perceive the area beyond the simulation map as free space, that is why figure 6 shows the free space beyond simulation boundary.

According to our analysis, when the map size decreases and number of agents are same or the number of agents increase,it leads to a bit of chaos in the environment which also leads to a lot of unsuccessful failures in the environment. For example, agents get stuck in the path because they can't see another agent and then the path planner even though showing a free path, fails to find path. This can be rectified by accessing the entities from the simulation environment. We are considering that in the future work. In a simulation of 20X20, we also found out that score decreases when number of agents increase because they sometimes start blocking each other or wait for other agent to attach and move. It is represented in Figure 4.

Implementation of path planning leads to efficient movement of agents in the simulation. We tested by checking the number of times agents reach the dispenser atleast once. For 5 agents, in a simulation size of 20X20 it takes on an average 40 timestamps to reach atleast one dispenser by each of the agents. But this speed increases drastically when there is only one agent and which also leads to highest score as shown in Figure 4.

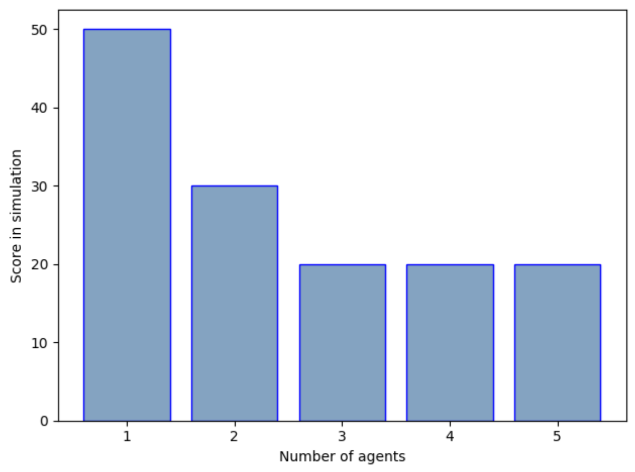

Fig. 4: Score for 100 simulation steps

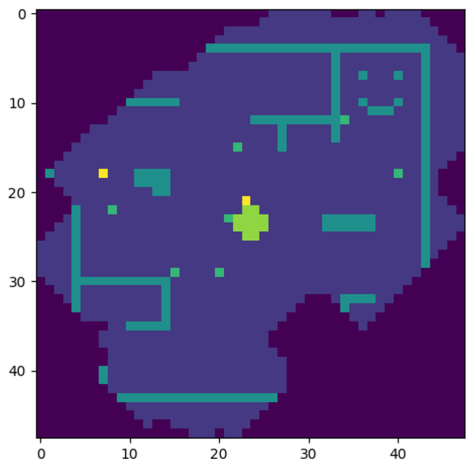

Fig. 5: Map merging in action on a big map.

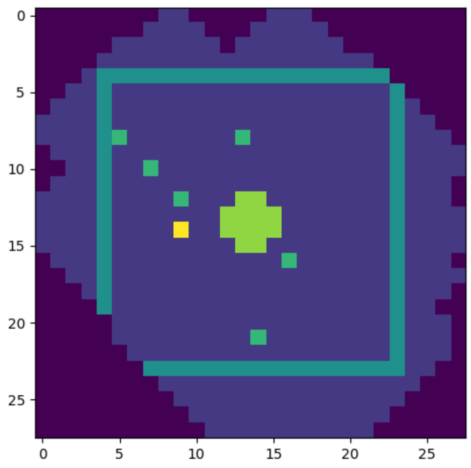

Fig. 6: Map Merging in action on a smaller map.

6 Future Work

For future work on this project, task allocation can be improved by have tasks split into sub-tasks based on their requirements (without any assumptions) and have agents collaborate on tasks assigned (attaching blocks to create more complex structures between them in order to complete tasks). Currently, agents only perform tasks with one block, but this could be improved and made possible for multiple blocks as well. Second, the communication protocol is not setup for task allocation. As the agents don't communicate about the tasks they are doing to other agents, the future version of task allocation would be to make it more centralized so that every agent can inform and update other agents about the task it is doing. This would make the task allocation system more robust and avoid any delays in decision making.

Improvements can also be made in exploration as at the moment, the strategy does not seem to result in a better movement plan than moving consistently north. Changing the threshold value makes the agent move in the other direction but this is still naive. A concrete strategy for exploration would be when agents go towards the wall that's closest to it, and not in one defined direction. Another improvement can also be in how long it takes an agent to find a dispenser before it starts submitting tasks. We tried to work on forcing agents to continue performing last action if the action fails but didn't have much success since it resulted in decision making time-out most of the times, which we intend on improving in the future.

Agents sometimes get blocked by another agents and take more time in waiting for other agent to move as they calculate the path, this can be improved by a better adaptive path planning technique. A future version of path-planner would be integrated with conflict based search which in real-time avoids 2 agents from crossing their paths. In our behaviour model, when two agents are near the dispenser, they sometimes attach to the block attached to another agent, we did not get enough time to make a robust behaviour model for our work. The behaviour model needs improvement which will enhance the score of agents.

7 Conclusion

The overall architecture is easy to understand and implement. The code is split into different modules, making code maintenance fairly easy. Subdivision into different modules makes it easier for debugging which for us in this project was one of the biggest challenge. The way RHBP planner is implemented is really modular as it gave us the flexibility to import any of the existing functions and behaviors without too much hassle. To conclude, the implementation of code in Python on top of ROS and RHBP gave us the freedom to implement a good system architecture.

Participation in the course to prepare for an entry into the MAPC Contest 2019, helped us to get an understanding of how to work with ROS and RHBP framework. This enabled us to plan and work on multi-agent systems, and also explore and interact with the given environment and scenario. The RHBP framework has powerful behavioural functions which can be adapted and tuned for each ageccordingly. We implemented the initial strategies needed for MAPC 2019 event on a theoretical level but we believe that it can be improved with wide research and testing .

References

1. Mapc 2019 contest. https://multiagentcontest.org/2019/
2. Mapc scenario documentation. https://github.com/agentcontest/massim$_2$019$/blob/master/docs/scenario.md$
3. Hart, P.E., Nilsson, N.J., Raphael, B.: A formal basis for the heuristic determination of minimum cost paths **2**, 7 (1968)
4. Hrabia, C.E., Lehmann, P.M., Battjbuer, N., Hessler, A., Albayrak, S.: Applying robotic frameworks in a simulated multi-agent contest. Annals of Mathematics and Artificial Intelligence (2018). DOI 10.1007/s10472-018-9586-x. URL https://doi.org/10.1007/s10472-018-9586-x
5. Hrabia, C.E., Wypler, S., Albayrak, S.: Towards goal-driven behaviour control of multi-robot systems. In: 2017 3nd International Conference on Control, Automation and Robotics (ICCAR), pp. 166–173 (2017)
6. Panait, L., Luke, S.: Cooperative multi-agent learning: The state of the art. In: Autonomous Agents and Multi-Agent Systems Volume 11, Issue 3, pp. 387–434 (2005)
7. Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., Wheeler, R., Ng, A.Y.: Ros: an open-source robot operating system **3**(3.2), 5 (2009)
8. Vidal, J.M.: Learning in multiagent systems: An introduction from a game-theoretic perspective. In: Alonso E., Kudenko D., Kazakov D. (eds) Adaptive Agents and Multi-Agent Systems. AAMAS 2002, AAMAS 2001, pp. 202–214 (2003)
9. Yoann, K., Philippe, M., Sébastien, P.: Everything can be agent! In: 9th International Conference on Autonomous Agents and Multiagent Systems, pp. 1547–1548 (2010)