A disaster scenario using multi-agent

Daniele Antunes Pinheiro [1] Débora Cristina Engelmann [2] Túlio Lima Baségio [3] Vagner Macedo Martins [4] Vinicius Lafourcade [5]

Abstract: Simulation of disaster scenarios helps to prevent real damage if an attack situation actually happens. When terror circumstances are upon us, human beings, we tend to panic e not think rationally, and that is where a simulation is useful. In this way, this article presents a simulation study in a disaster scenario on the campus of PUCRS in 2D and 3D map, where there are rescue points of victims, points of collection of supplies, hospital and agents.

***K*eywords:** Ros; Gazebo; AgentSpeak; Pyson; Jason; OpenStreetMap

1 Introduction

In a robotics project, regardless of whether there will be a physical application, the implementation of a simulation is fundamental. For that, the execution of tests with scenarios, environments, fixed objects, actions and real world situations are essential for getting good results.

Because of that, this work aims to create a simulated disaster scenario at the PUCRS' campus where some victims have been rescued and are receiving medical treatment in buildings. Two problems in this scenario require special attention: these buildings do not have enough supplies (medicines, food, water, blankets and curative) to properly care for the victims and some victims must be taken to the hospital for special treatment.

To attend this disaster scenario we are modeling a multi-agent simulation using ROS (to abstract the interaction with robots), a 3D scenario of the PUCRS' campus, a route simulation on the campus of PUCRS, AgentSpeak plans (to be used as the intelligence of the robots), Pyson and Jason.

This simulation will be composed of the following components:

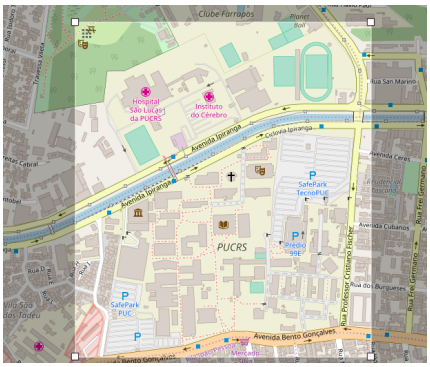
- A 3D map that should represent the PUCRS' campus (with its buildings, streets, etc.) to be used by the robots;
- A simulation of the route from one point to another of the campus using Rviz.
- ROS with Gazebo to abstract the interaction with the robots using the 3D map generated;
- Some plans using the AgentSpeak programming language to represent the robot's intelligence;
- Jason and Pyson to be used as bridge between ROS and AgentSpeak;

[1]Pontifícia Universidade Católica do Rio Grande do Sul, PUCRS
{daniele.pinheiro@acad.pucrs.br}
[2]Pontifícia Universidade Católica do Rio Grande do Sul, PUCRS
{debora.engelmann@acad.pucrs.br}
[3]Pontifícia Universidade Católica do Rio Grande do Sul, PUCRS
{tulio.basegio@acad.pucrs.br}
[4]Pontifícia Universidade Católica do Rio Grande do Sul, PUCRS
{vagner.macedo@acad.pucrs.br}
[5]Pontifícia Universidade Católica do Rio Grande do Sul, PUCRS
{vinicius.lafoucade@acad.pucrs.br}

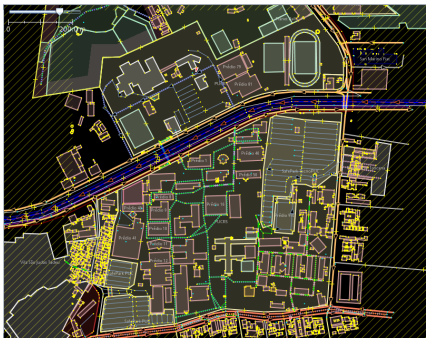
2 Scenario

The simulation aims to understand and learn the real world, without taking the risks and bearing the costs of the actual deployment. In order to create a real environment of the PUCRS campus, a scenario that simulates buildings and paths was constructed, as can be seen in Figure 1d. For this, the Gazebo simulator, which is a multi-robot simulator for complex indoor and outdoor environments, was used. It is capable of simulating the population of robots, sensors, and objects in a three-dimensional world. It generates both realistic sensor feedback and physically plausible interactions between objects (1).

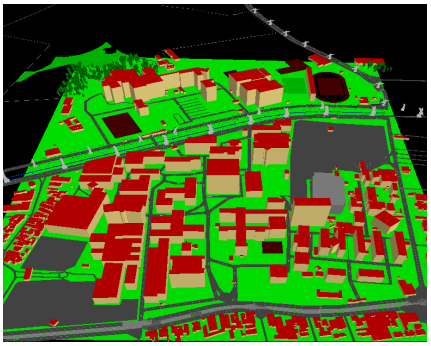
The scenario map was extracted from the OpenStreetMap site (2) which consists of a project to create and distribute geographic data for free to the world, as seen in Figure 1a. To select the area, the user fills the latitude and longitude ranges and then exports the file in .OSM format. You can edit the information or settings contained in the map, such as building height, street width, pavement type, position of objects on the map, or delete and add objects and information through the Java JOSM (3) application as seen in Figure 1b. After extracting or editing the map, the user can view it in 3D, using the OSM2World (4) Java application, which converts the map 2D to 3D, as seen in Figure 1c, another function of the application is to export the file for the .OBJ extension. With the OpenSceneGraph (5) graphical tool the user converts the .OBJ file format to .STL, which is compatible with the Gazebo simulator. After obtaining the map in .STL file format, the user initializes the Gazebo simulator and loads the map, as seen in the result in Figure 1d.



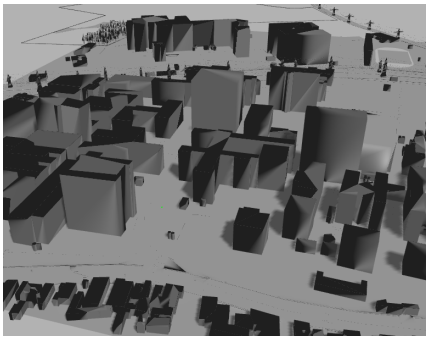
(a) Map extraction



(b) Map editing tool



(c) 3D map view



(d) Map of the Gazebo

Figure 1: Creating the Robotic Simulation Environment

Simulation of the navigation of a robot through the campus with Rviz

Rviz is a powerful visualization tool for ROS. It allows the user to view the simulated robot model, log sensor information from the robot's sensors, and replay the logged sensor information (6) (7). Rviz provides a configurable Graphical User Interface (GUI) to allow the user to display only information that is pertinent to the present task (6).

The Dijkstra algorithm solves the problem of the shortest path in a directed or non-directed graph with non-negative weight edges in computational time O ([e + v] log v) where (e) is the number of edges and (v) is the number of vertices. The Dijkstra algorithm does not solve problem in a graph with negative weights (8).

The simulation of the navigation of a robot the PUCRS campus was performed with the Rviz tool with the help of the Dijkstra algorithm. To do the simulation, the following steps were followed: cloning the MichalDoris/osm_planne repository; access the site map OpenStreetMap (2); change the robot_base_frame parameter from 'base_link' to 'map' from the osm_planner/config/ros_param.yaml file; create the route file with the .yaml extension inside the osm_planner/test folder defining the latitude and longitude coordinates of the source and destination; create the execution plan file with the .launch extension inside the 'launch' folder; remove the lines between 115 and 126 from the navigation_example.cpp file. After the adaptations in the project, run the application and observe the simulation of a robot running the least path, as seen in Figure 2.

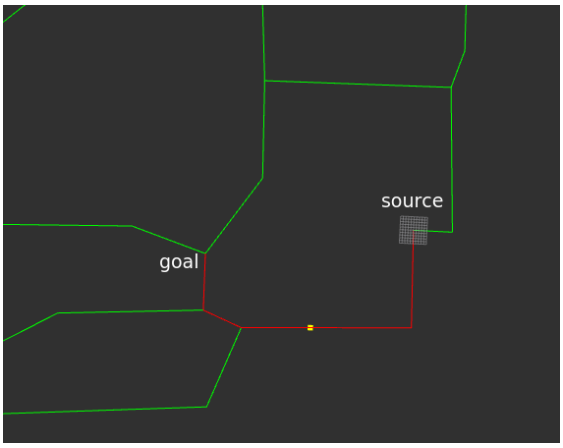


Figure 2: Result of robot route simulation in Rviz

3 AgentSpeak

AgentSpeak is an agent-oriented programming language, based on logic programming and the BDI architecture. This programming language was created by Anand Rao in 1996 and at the beginning was called as AgentSpeak(L) (9). AgentSpeak also was used as base to many other further works such as (10), (11) e (12). In the following sections we will describe some important concepts about AgentSpeak, the agents created and how each AgentSpeak plan was created.

As the name suggests, beliefs are elements that the agent assumes as truth, such as, the agent has a belief that indicates that the door is open. If another agent asks about the door state, the agent will answer that the door is open. Beliefs are represented by the symbol "+". A belief that corresponds to the

door open can be represented by +door(open). AgentSpeak allows 4 operation types on beliefs: addition, deletion, update and query. The addition, as demonstrated before, is represented by the symbol "+", the deletion by the symbol "-", the update by the symbol "-+" and query by the symbol "?".

Desires are elements that the agent desires to achieve (goal), e.g., the agent knows the door is open, but it desires the door to be closed, to achieve this goal in AgentSpeak we can use a desire !door(close).

Intentions are ways that the agent can try to use to achieve its goals, e.g., every time the agent wants to change the door state it demonstrates an intention to change that status. To represent this intention in AgentSpeak we can use +!door(Status), where the Status is a variable and in this case can be closed or open.

3.1 Agents Created

To meet the disaster scenario necessities we create two class of agents: building and transportation agents that will be described in the next sections.

The buildings are separated into three agent types: rescue point, collect point, and hospital.

- Rescue point: the rescue point agent will be responsible to request the supplies that it needs or to request a victim transportation to the hospital;
- Collect point: the collect point will be responsible to provide the supplies that the agent needs to delivery to the rescue point;
- Hospital: the hospital will be responsible to receives the victims transported by the agent;

Each building must be instantiated with a coordinate (X,Y) with a position in the 3D map, following the Jason or Python syntax and definitions.

The transportation is divided into two agent types: delivery agent and ambulance.

- Delivery agent: the delivery agent will be responsible to collect the requested supply and deliver to the rescue point.
- Ambulance agent: the ambulance agent will be responsible for taking the victim from the rescue point to the hospital.

Each delivery agent must be instantiated with its load capacity to transport the supplies, following the Jason or Python syntax.

3.2 Agents Workflow

Once the agent's definitions are completed, we started to create the AgentSpeak plans. In this section, we will present the agent's workflows created to meet the goals of the simulation suggested.

The buildings are very simple agents, except the rescue point that we will explain later, they are responsible only to provide their positions on the map to the transportation agents.

The rescue point is a more complex agent, it is responsible for request supplies or victim transportation. To choose which request it should make an aleatory number from 0 to 1 is generated as below:

- 0 to 0.17: Medicines

- 0.17 to 0.34: Food
- 0.34 to 0.51: Water
- 0.51 to 0.68: Blankets
- 0.68 to 0.85: Curatives
- 0.85 to 1: Victim transportation

When a supply is chosen, a new aleatory number from 0 to 1 is generated and multiplied to 100, this new number will be used to define the number of supplies the rescue point needs. When a supply is delivered, the rescue point requests a new one and this workflow continues until the simulation is interrupted.

To explain the agents' workflow of this class, we will start with the delivery agent. When a new supply request is available, the agent validates if its load capacity is higher than the number of supplies needed. If the quantity is higher than the load capacity of the agent, the agent discards this request and waits for a new one. If the agent can attend the request, it notifies the others delivery agents its intention to attends the request. To determine which agent will attend the request a simple validation of the agent name is made, the agent with the "lower" name has priority to attend the request, e.g., this validation with the agents "agent1" and "agent2", the "agent1" has priority. After that, the agent begins to move to the collection point where it will collect the supplies. When the agent arrives at the collect point, it begins to move to the rescue point to deliver the supplies. This workflow continues until the simulation is interrupted.

The ambulance agent is very similar to the delivery agent, but instead of wait for a new supply, this agent waits for a new victim transportation request. The heuristic to determine which agent has priority to attends the request is the same described previously about delivery agents. When the agent has priority to attend the request, begins to move to the rescue point, gets the victim when arrives and begins to move to the hospital. This workflow continues until the simulation is interrupted.

4 Pyson

The initial idea of this work was to use AgentSpeak (Py). Based on the AgentSpeak language, AgentSpeak (Py)[6] was developed in 2016 as a project at PUCRS. It is an interpreter for AgentSpeak, written in Python, to support an environment that allows multiple instances of agents. However, we found some limitations in AgentSpeak(Py), such as the representation of plans through literals that did not handle parameters and conditionals very well.

As a follow-up, we have used Pyson[7], another AgentSpeak interpreter in Python. This allowed us to use the same plans for both JSON and Python implementation, with one minor change: in Pyson, the method responsible for returning the ”go to" position needs to have a value returned.

Communication with ROS was done using RosPy[8]. In Python, we implement a *subscriber* that always receives X and Y positions of one of the robots in the scenario. When these positions are ready, AgentSpeak processes the map and decides where the robot should go. When AgentSpeak returns these directions, a *publisher* communicates with ROS and send the positions to the robot to walk. The figure 3 presents this system communication architecture.

[6]https://github.com/andrellsantos/agentspeak-py
[7]https://github.com/niklasf/pyson
[8]http://wiki.ros.org/rospy

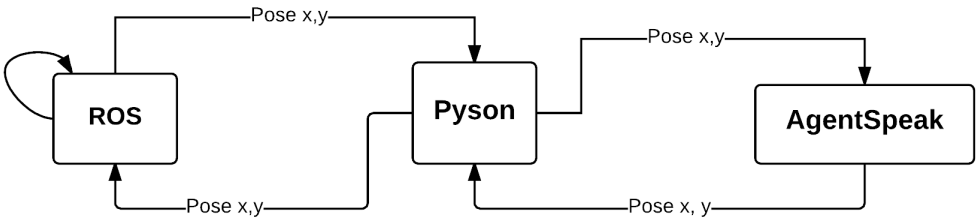


Figure 3: Communication architecture among Ros, Pyson and AgentSpeak.

Following the idea of a disaster inside the PUCRS campus, we need a robot with the capacity to carry a person or several supplements. So, we decided to implement the Grizzly[9] robot simulator. Grizzly is a robot designed for outdoor environment, used for military, mining and agricultural applications.

For the experiment, we launched two Grizzly robots in an empty scenario. Both robots were considered delivery agents, and they receive information about where they should go. The robots did not move at the same time. When one robot stops moving, the other one receives his directions and start to move. Figure 4 presents a screenshot from the simulation with the two Grizzly robots. The top of the figure shows our simulation with AgentSpeak, where the robots decide who is going to move to get supplies. In this simulation *robot1* moves first, as *robot2* informs he is busy.

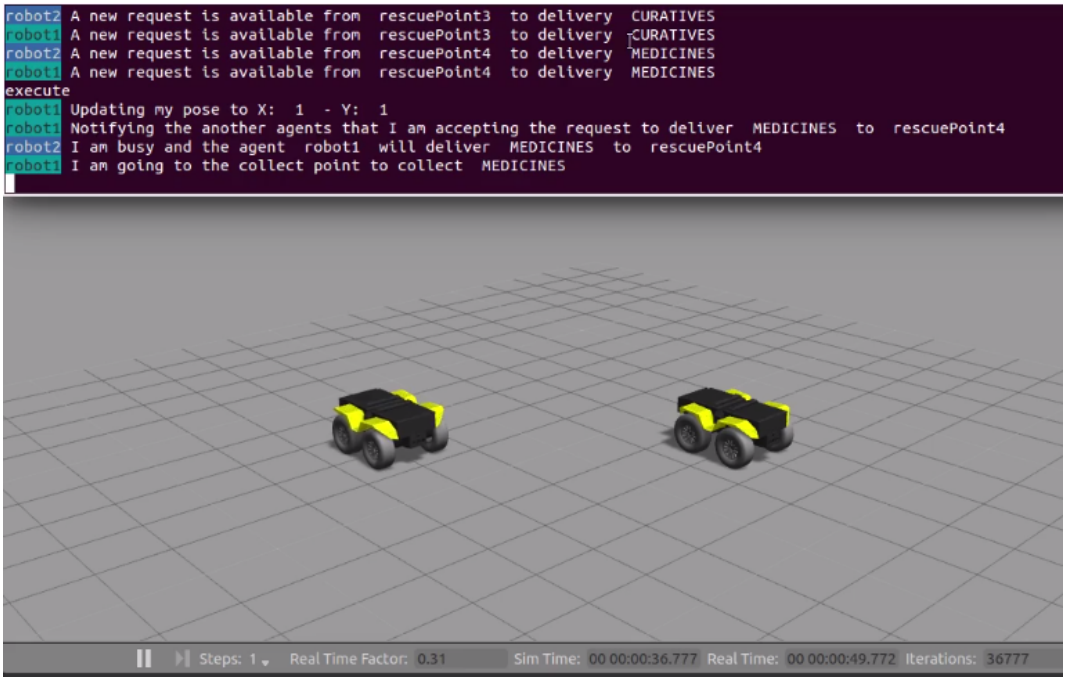


Figure 4: Simulation in Pyson with two robots.

[9]http://wiki.ros.org/Robots/Grizzly

During the development of this work, we have faced several limitations. The first one was the AgentSpeak. Before the start of this project, the language was unfamiliar and to learn it was a trick assignment. The attempt to use AgentSpeak(Py) was also a task that demanded some time and effort until we decided to change it for the use of Pyson. Pyson works with the plans created in AgentSpeak in a very similar way as Jason, which facilitated the implementation of the interface. And the last limitation, the integration with Ros, which consisted of launching two robots that should go towards the direction they were given. The movement of the robots did not achieve our expectations, as the robots didn't move correctly with the commands.

5 Jason

Another interpreter of the Agent Speak language that was used in this work is Jason (10), which is part of JaCaMo (13), a platform for the development of multi-agent systems. Jacamo is also composed by CArtAgO (14), which provides environment level programming based on shared distributed artifacts and, Moise (15), which provides organization level support for the agents. Figure 5 shows the three levels available in JaCaMo plataform.

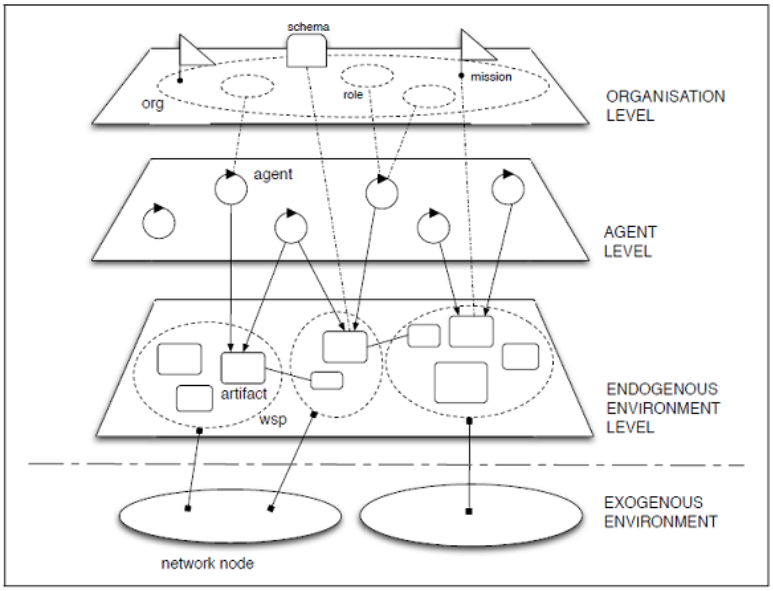


Figure 5: Overview of a JaCaMo multi-agent system, highlighting its three levels. (13)

The communication between JaCaMo and ROS was possible by using Rosbridge (16) protocol, which is a middleware abstraction layer that provides programmatic socket-based access to interfaces and robot algorithms provided by ROS. Rosbridge allows to publish and subscribe to ROS topics by sending JSON messages, provides support to service calls, setting up params, among other possibilities. Rosbridge is implemented in the rosbridge_suite[10] package that provides a WebSocket transport layer and is composed of other packages such as:

- rosbridge_library - responsible for converting the JSON to ROS commands and vice versa.

[10]http://wiki.ros.org/rosbridge_suite

- rosapi - provides service calls to getting ROS meta-information such as list of topics, services, params, among others.
- rosbridge_server - provides a WebSocket connection/interface to rosbridge.

The interface between Jason agents and the Rosbridge was implemented through an artifact in CArtAgO, which is based on Java programming language. In order to communicate our artifact with Rosbridge, we use the java_rosbridge library[11], which provides support for connection with Rosbrigde and publishing and subscribing ROS topics. Figure 6 shows the architecture overview of our solution.

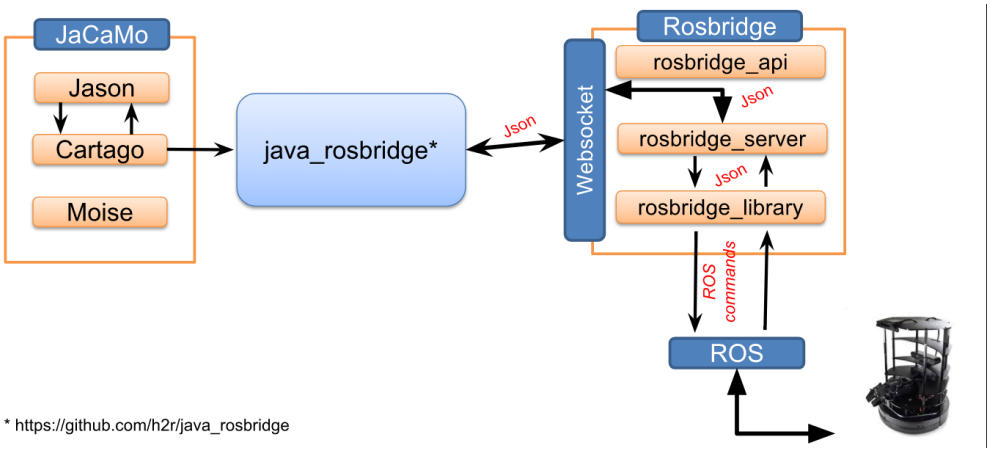


Figure 6: Architecture overview of our solution

In order to connect our artifact to java_rosbridge library we created an interface component. The interface component receives requests from an agent and forward the requests to java_rosbridge. When the java_rosbridge receives information (position, orientation, etc) from Rosbridge it sends the information to the interface component, which is responsible for updating agent's beliefs.

In our experiments we used Turtlebot[12] to check the technical feasibility of our architecture. In the experiments our agent subscribed to /odom topic (nav_msgs/Odometry messages) to get information about position an orientation of the Turtlebot and it publish linear and angular information to the /cmd_vel_mux/input/teleop topic (geometry_msgs/Twist messages) in order to move the Turtlebot around the environment. Figure 7 presents a screenshot of the simulation with the Turtlebot robot in Gazebo. The left side of the figure shows the JaCaMo console displaying messages about position, angles and goals.

6 Conclusion

This work showed a simulation for a disaster scenario inside the PUCRS campus. The idea was to have robots moving through the campus streets and carrying supplies and people to the Hospital. We couldn't achieve the simulation using the 3D map that was developed due to our lack of time and experience with ROS platform. This was one of the biggest challenges we face in this project, the knowledge from each participant of the group regarding ROS was not enough to cover all the things we need in this work. Other challenge was the size of the group that participate in this project. As each one was

[11] https://github.com/h2r/java_rosbridge
[12] http://www.turtlebot.com

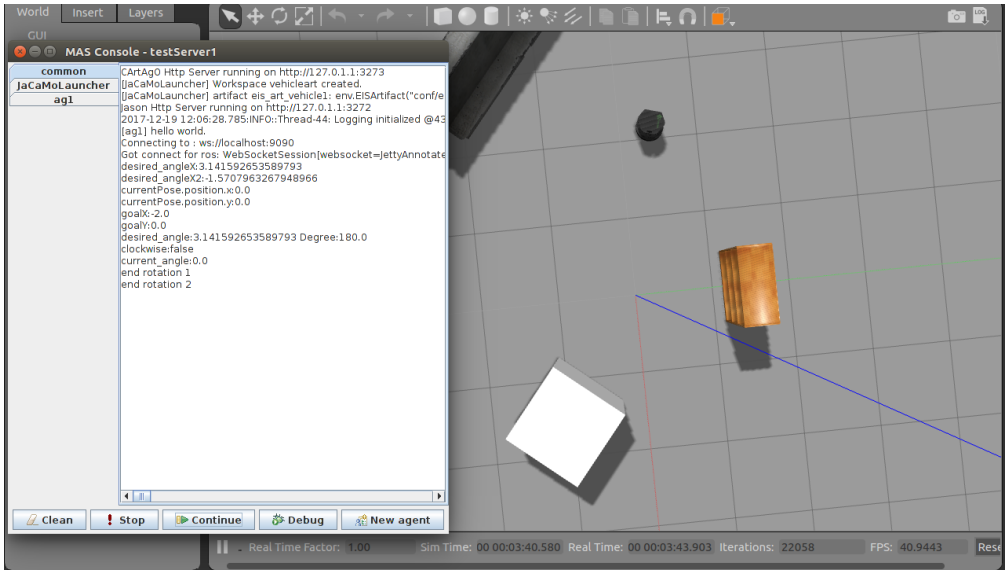


Figure 7: Simulation in Gazebo using JaCaMo

responsible for a particular piece there were many dependencies and that made it difficult to integrate the project as a whole. For future work, we propose the complete integration of all the parts that make up this work.

References

1 MAHTANI, A.; SANCHEZ, L.; FERNÁNDEZ, E. *Effective Robotics Programming with ROS - Third Edition*. [S.l.]: Packt Publishing, 2016. ISBN 978-1-78646-365-4.

2 OSM Open Street Map. Disponível em: <http://www.openstreetmap.org>.

3 JOSM Java Open Street Map. Disponível em: <https://josm.openstreetmap.de/>.

4 OSM2World. Disponível em: <http://osm2world.org/>.

5 OpenSceneGraph. Disponível em: <http://www.openscenegraph.org/>.

6 FAIRCHILD, C.; HARMAN, T. L. *ROS Robotics By Example*. [S.l.]: Packt Publishing, 2016. ISBN 1782175199, 9781782175193.

7 Rviz - ROS Visualization. Disponível em: <http://wiki.ros.org/rviz>.

8 ALGORITMO Dijkstra. Disponível em: <http://www.deinf.ufma.br/~portela/ed211_Dijkstra.pdf>.

9 RAO, A. S. Agentspeak (l): Bdi agents speak out in a logical computable language. In: SPRINGER. *European Workshop on Modelling Autonomous Agents in a Multi-Agent World*. [S.l.], 1996. p. 42–55.

10 BORDINI, R. H.; HÜBNER, J. F.; WOOLDRIDGE, M. *Programming multi-agent systems in AgentSpeak using Jason*. [S.l.]: John Wiley & Sons, 2007. v. 8.

11 BORDINI, R. H.; MOREIRA, A. F. Proving bdi properties of agent-oriented programming languages: The asymmetry thesis principles in agentspeak (l). *Annals of Mathematics and Artificial Intelligence*, Springer, v. 42, n. 1, p. 197–226, 2004.

12 VIEIRA, R. et al. On the formal semantics of speech-act based communication in an agent-oriented programming language. *Journal of artificial intelligence research.*, AI Access Foundation, v. 29, p. 221–267, 2007.

13 BOISSIER, O. et al. Multi-agent oriented programming with jacamo. *Sci. Comput. Program.*, Elsevier North-Holland, Inc., Amsterdam, The Netherlands, The Netherlands, v. 78, n. 6, p. 747–761, jun. 2013. ISSN 0167-6423. Disponível em: <http://dx.doi.org/10.1016/j.scico.2011.10.004>.

14 RICCI, A.; PIUNTI, M.; VIROLI, M. Environment programming in multi-agent systems: an artifact-based perspective. *Autonomous Agents and Multi-Agent Systems*, v. 23, n. 2, p. 158–192, 2011. Disponível em: <http://dblp.uni-trier.de/db/journals/aamas/aamas23.html\#RicciPV11>.

15 HANNOUN, M. et al. Moise: An organizational model for multi-agent systems. In: ____. *Advances in Artificial Intelligence: International Joint Conference 7th Ibero-American Conference on AI 15th Brazilian Symposium on AI IBERAMIA-SBIA 2000 Atibaia, SP, Brazil, November 19–22, 2000 Proceedings*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000. p. 156–165. ISBN 978-3-540-44399-5. Disponível em: <https://doi.org/10.1007/3-540-44399-1_17>.

16 CRICK, C. et al. Rosbridge: Ros for non-ros users. In: ____. *Robotics Research : The 15th International Symposium ISRR*. Cham: Springer International Publishing, 2017. p. 493–504. ISBN 978-3-319-29363-9. Disponível em: <https://doi.org/10.1007/978-3-319-29363-9_28>.