Platypus Boat Documentation

Release 0.1

Alexandre Amory

Apr 22, 2019

Contents

1 About the Campus **1**

2 Summary **3**

2.1 Building the PUCRS Campus on Gazebo . . . 3

2.1.1 1. Extraction of campus map PUCRS . . . 3

2.1.2 2. Map editing . . . 3

2.1.3 3. 3D Map Creation . . . 4

2.1.4 4. Conversion from *.obj* format to *.stl* format. . . . 4

2.1.5 5. Upload the map in the Gazebo . . . 4

2.1.6 6. Loading Turtlebot in the New World . . . 4

2.2 Supported Robots . . . 4

2.2.1 Drone . . . 5

2.2.2 Small Car . . . 5

2.2.3 Big Car or Truck . . . 5

2.3 Setting Up Jason . . . 5

2.4 Setting Up Pyson . . . 6

2.4.1 1. Install Pyson . . . 6

2.4.2 2. Download the Grizzly robot . . . 7

2.4.3 3. Add the launch file from to Grizzly . . . 7

2.4.4 4. Copy the package node . . . 7

2.4.5 5. Lauch Grizzly . . . 7

2.4.6 6. Run Pyson . . . 7

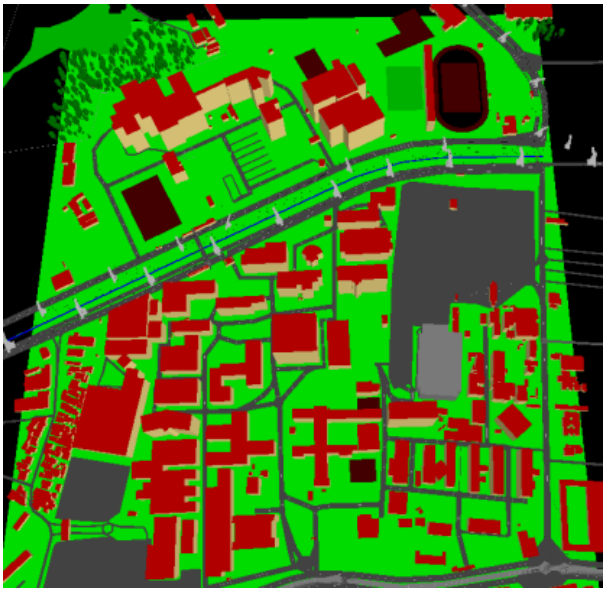
2.5 AgentSpeak Codes . . . 8

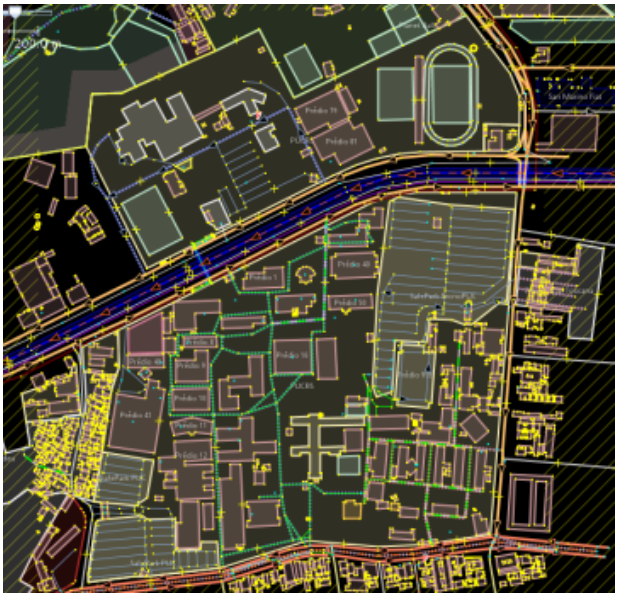
2.6 LSA Contributors . . . 9

3 Feedback **11**

CHAPTER 1

About the Campus





Place here some robots talk a little bit about the purpose of this document.

Show some youtube videos demonstrating its usage.

CHAPTER 2

Summary

2.1 Building the PUCRS Campus on Gazebo

Creation of the PUCRS Campus Scenario in the Gazebo

To run the example we created, you need:

1. Ubuntu 16 with Kinect distribution installed (ros-kinetic-desktop-full)
2. Python

2.1.1 1. Extraction of campus map PUCRS

Go to the website Open Street Map.

Click the **Export** button at the top of the screen.

Fill in the latitude and longitude coordinate range (scroll through the coordinate fields by pressing a **Tab** key to view an area on the map).

Example (EPSG:4326 - WGS84):

left latitude = -51.1839

right latitude = -51.1635

upper longitude = -30.0505

lower longitude = -30.0615

Click the **Export** button the left side of the screen (the file extension will be *.osm*).

2.1.2 2. Map editing

Visit the site *https://josm.openstreetmap.de/wiki/Download* to download the *JOSM <https://josm.openstreetmap.de/wiki/Download>_* application *josm-tested.jar*.

Open an application with the command:

$ java -jar josm-tested.jar

Choose the *.osm* extension file.

Changes to map attributes are made in the right part of the application.

Save the changes

2.1.3 3. 3D Map Creation

Access the website *http://osm2world.org/* to download the OSM2world application.

Open an application with the command:

$ java -jar OSM2World.jar

Check out the changes performed in the previous step.

Export the file to the object format (*.obj*).

2.1.4 4. Conversion from *.obj* format to *.stl* format.

Documentation: *http://www.openscenegraph.org/index.php/documentation/guides/user-guides/55-osgconv*

Run the command:

$ osgconv map_pucrs.obj map_pucrs.stl

2.1.5 5. Upload the map in the Gazebo

Run the command:

$ gazebo to load the empty environment.

Press **Ctrl + M** or go to the **Template Editor**.

Click **Add Custom Shapes**.

Select the *.stl* extension file.

Save the template.

Close the **Template Editor**.

Save the world file.

2.1.6 6. Loading Turtlebot in the New World

Run the command:

$ roslaunch turtlebot_gazebo turtlebot_world.launch world_file:=/opt/ros/kinetic/share/turtlebot_gazebo/worlds/campus.world

You will see a 3D scenario in the Gazebo Simulator and a Turtlebot at the point of origin.

2.2 Supported Robots

at least one of each type must be supported. describe the main topics used to control each robot.

2.2.1 Drone

To be done. Perhaps Hector Quadrotor.

2.2.2 Small Car

Grizzly simulator.

sudo apt-get update

sudo apt-get install ros-indigo-grizzly-simulator ros-indigo-grizzly-desktop ros-indigo-grizzly-navigation

roslaunch grizzly_gazebo grizzly_empty_world.launch

2.2.3 Big Car or Truck

To be done.

2.3 Setting Up Jason

- JaCaMo (http://jacamo.sourceforge.net/)

Is a framework for Multi-Agent Programming that combines three separate technologies, each of them being well-known on its own and developed for a number of years so they are fairly robust and fully-fledged. JaCaMo is a combination of Jason, Cartago and Moise.

- Jason (http://jason.sourceforge.net/wp/)

Is an interpreter for an extended version of AgentSpeak. It implements the operational semantics of that language, and provides a platform for the development of multi-agent systems, with many user-customisable features.

- CArtAgO (http://cartago.sourceforge.net/)

Is a general purpose framework/infrastructure that makes it possible to program and execute virtual environments – also said virtual / application / software environments – for multi-agent systems.

- Moise (http://moise.sourceforge.net/)

Is an organisational model for Multi-Agent Systems based on notions like roles, groups, and missions. It enables an MAS to have an explicit specification of its organisation. This specification is to be used both by the agents to reason about their organisation and by an organisation platform that enforces that the agents follow the specification. To install JaCaMo Eclipse Plugin please go to http://jacamo.sourceforge.net/eclipseplugin/tutorial/

- Rosbridge (http://wiki.ros.org/rosbridge_suite)

For communication between Jason and ROS was used the Rosbridge. It provides a JSON API to ROS functionality for non-ROS programs. There are a variety of front ends that interface with rosbridge, including a WebSocket server for web browsers to interact with. Rosbridge_suite is a meta-package containing rosbridge, various front end packages for rosbridge like a WebSocket package, and helper packages. Allows publishing or subscribing to ROS topics by sending a JSON. Covers also service calls, getting and setting params, and more.

Divided into three parts:

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

rosapi: provides service calls for getting ROS meta-information list of topics, services, params, etc.

rosbridge_server: provides a WebSocket connection/interface to rosbridge.

To install Rosbridge run:

```
` $ sudo apt-get install ros-<rosdistro>-rosbridge-server `
```

Initialize RosBridge:

```
` $ source /opt/ros/<rosdistro>/setup.bash `
```

```
` $ roslaunch rosbridge_server rosbridge_websocket.launch `
```

In other console run the tortoisebot

```
` $ roslaunch tortoisebot tortoisebot.launch `
```

To see what is being published in the topic run in other console

```
` $ rostopic echo /cmd_vel `
```

- Import the project in eclipse

Run JaCaMo Application testServer1.jcm

Your turtlebot will receive information from the agent programmed in Jason regarding the move goals and will send information about your current position and orientation to the agent.

This information can be viewed on the JaCaMo console that will open when the simulation starts.

2.4 Setting Up Pyson

Pyson is a JASON-Style AgentSpeak interpreter for Python.

Currently this plugin is under development but we could not found the currently version.

To run the example we created, you need:

1. Ubuntu 14.04 with Indigo distribution installed.
2. Python

And should follow this steps:

1. Install Pyson
2. Download the Grizzly robot
3. Add the launch files from to Grizzly
4. Copy the package node
5. Lauch Grizzly
6. Run Pyson

2.4.1 1. Install Pyson

To install Pyson is simple. First, download Pyson source code from the GitHub repository. Then, you need to execute in a terminal the command:

```
sudo python setup.py develop
```

This command will install all the Pyson's dependencies.

2.4.2 2. Download the Grizzly robot

For this work, we use Grizzly simulator. You need to download the source from GitHub and place it in *src* in your *catkin_ws*. After that, run:

catkin_make

And

source [YOUR PATH]/catkin_ws/devel/setup.bash

2.4.3 3. Add the launch file from to Grizzly

We created an odometry node to get the robot position. You need to replace the file at *catkin_ws/src/grizzly_simulator/grizzly_gazebo/launch/base_gazebo.launch* for the file at */launch/Grizzly/base_gazebo.launch*. The file on this repository already contains the node to get the positions information:

<node pkg="check_odom" type="check_odometry" name="main.py" output="screen" >

Where:

- The pkg is the name of the package.
- The type is the name of the node described at our script
- The name is the name of our script
- Output is the way the information is printed

For the example, we launched two robots. So, you also need to replace the file at *catkin_ws/src/grizzly_simulator/grizzly_gazebo/launch/grizzly.launch* for the file at */launch/Grizzly/grizzly.launch*.

After you replace the file, run

catkin_make

2.4.4 4. Copy the package node

In this repository, inside src/pyson/ there is a folder named *check_odom*. You need to copy this folder to your *catkin_ws*.

2.4.5 5. Lauch Grizzly

To launch Grizzly, run:

sudo apt-get update

sudo apt-get install ros-indigo-grizzly-simulator ros-indigo-grizzly-desktop ros-indigo-grizzly-navigation

roslaunch grizzly_gazebo grizzly_empty_world.launch

2.4.6 6. Run Pyson

Open other terminal and run:

roscore

In other terminar, enter the folder *check_odom* you just copied to your catkin_ws and run:

python main.py

You will se the plans and the actions on your console, and the robot moving on Gazebo.

2.5 AgentSpeak Codes

To test the AgentSpeak plans created to this work we have two options, with JASON or with Pyson (another tools or libraries were not tested here).

- **JASON**
  - To test with JASON, the JASON plugin must be installed on the Eclipse (more information at http://jason.sourceforge.net/mini-tutorial/eclipse-plugin/#id.iyxfqy5c7qw)
  - After the plugin is installed, we can create a new JASON project. The AgentSpeak files (.asl files) must be placed at the folder /project_name/src/asl
  - To add agents to the environment, we need to change the [project_name].mas2j file. In this file we have a section called "agents" and at this section we can add a new building agent as the following: collectPoint1 collect_point [beliefs="pose(7,8)"]; This example is adding a new agent called "collectPoint1" using the collect_point ASL file and starting it with a initial belief called "pose" with two parameters "7" and "8". For delivery agents, we need to add as following, where we need to provide a load capacity: deliveryAgent1 delivery_agent [beliefs="loadCapacity(100)"];
  - After the agents are added to the environment, we can start the project as default by Eclipse (a play button placed at the top of the Eclipse).
  - To test this scenario, we created a executor_agent that is responsible to call the execute belief of the transportation agents in a loop.
- **Pyson**
  - To test with Pyson, the Pyson plugin must be installed (more information at https://github.com/niklasf/pyson/)
  - After the plugin is installed, we can execute our AgentSpeak plans in every one folder.
  - To test it, we need to create a new .py file following the example.py file placed at ../../../src/agentspeak/Pyson
  - After that, we need only to execute in a terminal the command python [file_name].py to test it.
- **Plans created:**
  - To achieve the goal of this work we have created some AgentSpeak plans. Initially, all agents start only printing their basic information, like initial position and initial status.
  - After that, the rescue points start sending their position to the delivery agents and ambulances and then send a new supply request or a victim transportation request;
  - When a delivery agent receives a new supply request, the agent validates the supply quantity through its load capacity. If the supply quantitiy is lower than its load capacity, the agent add this request to its belief base;
  - At this moment, the rescue points change their status to "WAITING";
  - Now, the delivery agent will notify all the other agents that it will do the request. The another agents will receive this broadcast and the agent with the lower name will be able to do the request, e.g., the agent1 is notifying the agent2 about the request and in this case the agent1 has priority about the agent2.

  - When the agent accepts a request and can do that, it changes its status to "GOING_TO_COLLECT" and asks the bridge (Pyson or JASON) to move the robot at ROS to the position of the collect point.
  - When the agent arrives the destination, it collects the supply and changes its status to "GOING_TO_DELIVERY_POINT", then the agent asks the brigde to move the robot at the ROS to the position of the rescue point that asked the supply.
  - When a supply is delivered, the building agent changes its status do "AVAILABLE" and in the next execution it will ask for a new supply.
  - The process described above about the delivery agent is similar to the ambulance agent, the difference is instead of collect a supply in the collect point, the first step after receive a new request is to get the victim at the rescue point.

- **Limitations**
  - We found a limitation on Pyson that we cannot update a belief using "-+" operator. For this reason we created simple plans only to remove the current belief and add a new one with an updated value.
  - The conditional on the AgentSpeak-py never satisfied our plans while in Pyson we could achieve the conditions properly.

2.6 LSA Contributors

The list of contributors to this document.

- Daniele Pinheiro
- Debora Cristina Engelmann
- Vinicius Lafourcade
- Vagner Macedo Martins
- Túlio Basegio

CHAPTER 3

Feedback

Don't hesitate to ask about some additional info or next guides and also if you find some mistakes, please let me know. This can be done by submitting an issue or a push request on github.