ROS-Explorer: a ROS module that implements harmonic potential fields based exploration

Atila Leites Romero
Programa de Pós-Graduação em Ciência da Computação
Pontifícia Universidade Católica do Rio Grande do Sul
Porto Alegre, Rio Grande do Sul
Email: atilaromero@gmail.com

Abstract

This paper describes a work done during the 2018 class Robótica Móvel Inteligente (Intelligent Mobile Robotics) under supervision of Prof. Alexandre de Morais Amory, Prof. Renan G. Maidana, Prof. Roger Leitzke Granada and Prof. Vitor A. M. Jorge.

In this class, a ROS module was developed to automate robot exploration, where a map is build using laser readings and movements commands.

Keywords – ROS, exploration, harmonic potential fields, path planning

1 Introduction

The developed project address two problems faced during the class beginning: installation of ROS[3] and creation of a map. Installation of ROS may not be straightfoward if the user does not use the recommended Ubuntu version. And creation of a map in ROS by manually moving the robot inside a Gazebo[2] simulation can be tedious. For the first problem, a properly configured docker container, as the one provided in the project repository, can be an easy solution. For the second problem, the ros-explorer module can be used, as it automates the exploration and map creation process.

The project source code is available at http://github.com/atilaromero/ros-explorer

This paper is structured as follows: section "Docker" describes how using docker with ROS can be beneficial, section "Laser mapping" discusses how the laser readings were used to create maps, section "Harmonic potential fields" give a brief introduction on the subject and details how the technique was implemented to perform path planning, section "Map publish" describes how the generated map is published to ROS, and section "Results" discusses achivementes, problems and future work.

2 Docker

Docker[1] is a technology that creates container based environments. It has some similarities with virtual machines, but without the performance loss. It may be also compared to chroot solutions, but its usage is more robust and simple.

A running docker container creates a process that runs directly in the host's kernel, but it has its own filesystem.

Docker allows, for example, to run Ubuntu 16.04 on a Kali distribution, which was the setup used during development of this work. For ROS[3] users this may be very practical, since ROS installation instructions are based on specific Ubuntu versions. Without docker, a ROS user with the wrong linux distribution or version has to choose between compile ROS from source or to reinstall his operating system. With docker, the same user can keep his host operating system intact, use the recommended Ubuntu[5] version inside the ROS container and even have different ROS versions in different containers.

One drawback of using docker is that the container preparation can consume some time, as any operating system install procedure does, but specially regarding graphical cards because it is often necessary to install inside the container the same version used in the host, which is a step that cannot be automated since it is host specific.

For the kinetic version of ROS, which recommends Ubuntu 16.04, this work is already done, except for the graphical card part, in the Dockerfile provided at the ros-explorer[4] repository (http://github.com/atilaromero/ros-explorer).

The usage instructions are also on the repository, but it essentially consists in downloading a docker image of about 1GB using "docker pull" and then using "docker run".

Although beneficial, the ros-explorer module is not required to run in a docker environment,

as it will function in the exact same way in a traditional physical instalation. Also, the provided docker image is not specific to the ros-explorer package, as it may be used with other packages too.

3 ROS-explorer usage

Installation of the ROS-explorer module can be done by clonning the project repository under the catkin workspace source directory, and then by calling "catkin_make" under the catkin workspace root.

Assuming a simulation is already running, which can be accomplished by calling "roslaunch turtlebot_stage turtlebot_in_stage.launch", the explorer program can be starting calling "rosrun ros-explorer explore.py".

The only input used from ROS to perform exploration are the laser readings, gathered at the "/scan" topic. The position of the robot is infered from the motion commands sent to the wheels at the /cmd_vel_mux/input/navi topic. The other two outputs are the generated map, published at the "/mymap" topic, and the potential field, published at the "/myfield" topic.

4 Laser mapping

The laser readings received from ROS are an array of numbers, each representing the distance until an obstacle at a particular angle from the robot front. The message containing this data also informs the range of the angles and angle increment from one array element to the next.

When the laser does not detect any obstacle in the laser range, it return a NaN as the reading to that angle.

To convert these 1D readings to a cartesian map, first the readings are converted to a polar 2D representation. One axis is the angle, going from $-\pi$ to π and using angle increment as resolution. The other axis is the distance, going from zero up to the laser range, which were 10 meters during simulation tests, using 20 centimeters as resolution.

In this representation, obstacles are marked with value 1, free areas with value 0, and unknown areas with value -1. Areas before an obstacle are considered free, and areas behind obstacles or for which no laser reading is available are considered unknown.

A polar to cartesian transformation is then applied, producing a visual meaningful map of a laser reading. Using the estimated position for the robot, the latest laser readings map can be combined with the current worldmap, which begins empty.

This is accomplished by summing them together: if a location is marked as empty (-1) in one map and occupied (+1) in other, results in unknown (0); if both maps classifies the location the same way, the result maintains the classification; if one of the maps classifies the locations as unknown (0), the other's map value prevais.

In order to keep the worldmap accumulated reading values between -1 and +1, an attenuation function is applied on the result:

$$f(x) = \frac{sin(\alpha x)}{\alpha} . \frac{\alpha\beta}{sin(\alpha\beta)} \tag{1}$$

This function has the following interesting properties:

- $f(0) = 0$: unstable equilibrium on 0
- $f(\beta) = \beta$: stable equilibrium on β
- when $|x| < \beta$ then $f(x) > x$: it amplifies small values
- when $|x| > \beta$ and $|x| < \pi/\alpha$, then $f(x) < x$: it decreases slight high values

Those claims come from the position of the inflection point, which occurs when $f(x) = x$:

$$f(x) = x \tag{2}$$

$$\frac{f(x)}{x} = 1$$

$$\frac{\frac{sin(\alpha x)}{\alpha} . \frac{\alpha\beta}{sin(\alpha\beta)}}{x} = 1 \quad \text{(by equation 1)},$$

$$\frac{sin(\alpha x)}{\alpha x} = \frac{sin(\alpha\beta)}{\alpha\beta}$$

$$x = \beta \tag{3}$$

After some experimentation, the values $\alpha = 1.1$ and $\beta = 0.6$ were chosen. In fact, the whole formula was found in an empirical way, by trial and error. Only after it was already working that an explanation was developed. Substituting the given values in the formula, it simplifies to:

$$f(x) = 0.9786 sin(1.1x) \tag{4}$$

The applied solution requires one reading to overwrite the worldmap, and the choosen parameters are such that $|f(x)| < 1$ is always true, which can be verified by using the largest possible input for x, which is 2 (two consecutive +1 readings): $f(2) = 0.7912$.

In pratice, this means that the current reading prevails over previous ones, that the values are always between -1 and +1, and that the update procedure has only two steps: sum the current readings with the worldmap and apply the attenuation function.

5 Harmonic potential fields

In this technique, obstacles and objectives are marked in the map as extreme opposing values. For that reason, a potential field map has to use a different set of values than the ones described in the previous section. Keeping the walls values at +1, the value for unknown areas is set to -1 and the value for free areas is set to 0, which is the opposite of what was done before.

The harmonic potential fields technique is incremental: at each step, the value of each cell is calculed as the gradient of the current region, using the four nearest neighbours values. But there is an exception, cells that are walls or objectives do not change values. That way they work as sources or sinks. After enough steps, finding a path to an objective can be done by simply following the lowest potential from the current position.

The first attempt was performed using a for-loop construction. But, being done in python, the performance of the algorithm was not satisfactory. A better approach was tried next, using openCV to apply the following convolutional kernel to the map:

0	0.25	0
0.25	0	0.25
0	0.25	0

For the unchangeable values, instead of avoiding them, they were overwritten at the convolution and overwritten again to restore their original values.

Despite using more operations, this approach performed better because it uses openCV instead of pure python. Later the -1 value was substituted for -1000, which made the path discovery faster.

6 Map publish

The worldmap constructed by joinning the laser readings together is being published on the topic "/mymap", and the potential fields map on the topic "/myfield". This allows to visualize the maps on rviz while they are being made.

But the since the only source of inputs of ros-explorer comes from the laser readings, the only common frame of reference between the map being built and the map already known by rviz is the robot itself.

So, after publishing each of the maps on their topics, ROS must be informed on how they must be placed in relation to some reference, in this case "base_link", the base of the robot.

This is done by using a coordinate system transform that puts the origin of the map below the robot position, facing forward. Thus, as the robot moves, the published map moves below it in the opposite direction, staying always in the same place in the floor frame of reference.

In rviz this showns as a map that is just in the right place.

7 Results

The module has succeded at mapping an simulated environment, but it assumes that eventual errors between commanded movement and execution are small. In a real scenario it may be necessary to add a particle filter or other similar technique to correct the robot position.

An known limitation of the module is the size of the map, always fixed at 400x400 pixels. A dynamic sized map, or at least a configurable one, is the most important improvement pending.

The path planning using harmonic potential fiels was successful, but the limited size of the map imposed a problem, as the robot could not perform exploration well when its laser range reached a border.

References

[1] Docker. https://www.docker.com/, 2018. [Online; accessed 16-December-2018].

[2] Gazebo. http://gazebosim.org/, 2018. [Online; accessed 16-December-2018].

[3] ROS. http://www.ros.org/, 2018. [Online; accessed 16-December-2018].

[4] ros-explorer. https://github.com/atilaromero/ros-explorer, 2018. [Online; accessed 16-December-2018].

[5] Ubuntu. https://www.ubuntu.com/, 2018. [Online; accessed 16-December-2018].