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

Atila Leites Romero

December 14, 2018

Abstract

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

1 Introduction

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.

The developed project address two problems faced during the class beginning: installation of ROS 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 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 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 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 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 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:

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

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:

$f(x) = x$

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

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

After some experimentation, the values $\alpha = 1.1$ and $\beta = 0.6$ were chosen, giving after simplification:

$f(x) = 0.9786 sin(1.1x)$

This approach was adopted as a temporary replacement to Hidden Markov Models (HMM), but it was never changed because its results were satisfatory. HMM is less sensitive to changes, requiring more readings to change a value, while the applied solution requires one or two readings. In pratice, this means that the current reading prevails over previous ones. When using in a real scenario HMM may be a better choice, but in a simulation the simplicity and speed of this solution are helpful.

5 Harmonic potential fields

6 Map publish

7 Results

References