

Gazebo World Generator

12/10/2021

**URAS ALKAŞ**

**Table of Contents:**

Introduction..............................................................................................................................3

Language and Libraries used...................................................................................................4

Explanation of the code file by file..........................................................................................5

Result and Future Work...........................................................................................................6

**Introduction:**

When writing software for a robot, Milvus Robotics (like the rest of the world) uses a framework (not sure if that is the right term) called ROS(Robot Operating System). In ROS we basically build the software with each task inside different modules called nodes. Along this easy to implement structure ROS has many pre made open source packages distributed by the ROS community and it has many plugins so we can make our ROS programms run together with other software to create even more sophisticated systems.

One of such plugins is the ROS-Gazebo plugin. Gazebo is basically a simulation programm to test our robot models by running the written software inside a loop where the outside parameters are simulated by Gazebo itself.

In order to simulate our robots we first need to create a simulation world with walls and such . Our world should model the real world where our robots will work (mostly warehouses and factories) as realistic as possible so we can make better decisions for change in software as we simulate the performance of our robots in these worlds.

One way to create these worlds could be to generate the world from the images of the factory mapped by the robot traversing every corner of the workplace with its lidars attatched. This way we could get (atleast geometrically) the complete copy of the factory as a simulation world.

The project I worked on during the time I interened at Milvus was the create a program to create these simulation worlds or we could say a programm to generate the world from a binary image where the colour white is ground while the colour black is the obstacles or walls. Also in this programm the user can add custom models and the programm has to create another binary image with the custom models are also added as obstacles in the newly generated map image.

**Language and libraries used:**

The only programming language used in this project is C++. But since the model and world files are all created as sdf files(which is basically XML), the maintainer or reader of this code should be familiar with the XML markup language.

* OpenCV is used to process the map image data.
* Tinyxml2 is used to parse XML.
* Eigen library is used to handle the linear algebra operations to process the transformations the added models undergo whith the given pose vectors.

I added no opensource code to this project all of it is written in my computer in Milvus Robotics headquarters.

**Explanation of the code file by file:**

Main Function:

The main function first reads the image and flips the image to make it accomodate with the positive y axis understanding of normal poeple(like gazebo) converts it to a binary image. Then it takes the scale value from the user. This scale is well explained in the read me file of the programm. Then it generates the borders(which are basically rectangles with given top left and bottom right corners coordinates) with the generate border function from the compose\_rectangles.cpp file which returns these coordinates as pointer to pointers of integers (can be tought as 2d arrays where each row of the array consisting of 4 integers where first two integers are coordinates of the top left corner and the other two are bottom right corner coordinates ).

generate\_borders function does all of these by scanning the image line by line and when it finds a line where the black pixels are all on the same row with the previous lines section of black pixels then the rectangle which is the owner of that previous section is extended such that it also covers this new section of black pixels.

Returning to the main function we call the parser\_func from the xml\_parser.cpp file to create a model (a sdf file) of walls using the coordinates of rectangles obtained previously. For each wall(basically a box created inside the sdf file) we also create a Box object to put the wall inside the final map image by call this objects put\_map method. The implementation for all the model(box sphere mesh etc.) classes is inside the model.cpp file. For each wall the parser\_func function makes the generation using the generateWall function in the same file. The caster function inside this file is used to create char pointers from the strings like pose so functions like SetText can be fed with the data type they require.

This time the main function calls the generator function from the word\_generator.cpp file whose job is to parse another sdf file which will be our world file. To this world file it adds the previously created wall model which had the name “new\_model” and the all the other custom model that it takes from the user which should be inside the model file of gazebo. It uses the generate\_map function from the map\_png\_generator.cpp file to add these models to the final map image. It also creates light on the first day of creation by adding the sun model from the model file.

The generate\_map function basically checks for the type of model first then creates then creates a model object according to the type using the classes in models.cpp file. Then using these objects put\_map method it adds the shape to the map image. This file has 3 other functions to help the generation process.

xml\_iterator to find the required element in the xml files.

compare\_string\_with\_ptrtochar to check if that string is the same as that char pointer

string\_parser to change a char pointer to a char pointer to pointer by splitting the char pointer by the spaces.

models.cpp as previousl told is the implementation file for all the model classes. It has the base shape class and the classes box, sphere, cylinder and mesh inherited from the shape class. These classes have all the methods to put these objects on the map image. Sometimes they got a little confusing to handle processes like what the box would look like when transformed by the pose given by the user and such. Still it handles them mostly well so we can see the final models on the map image.

After all these the main method flips the image back and saves it as result.jpg to the project folder.

**Result and Future Work:**

Altough the program more or less does everything required by it the main problem with the current version is that when reading the initial image it tries to analyze the whole image in to rectangular sections. This obviosly creates a greate number of walls for non rectangular image parts and makes it hard for gazebo to process wich gives us poor simulation performance. First improvement that can be made is to process the image for every type of shape possible. Circular parts can be made out of cylinders and non geometric parts of the image can be made out of mesh objects. Other than that the mapping of the cylindirical models to the image is yet uncomplete since the program does not handle orientation of cylindirical models yet.

No referances since I used no code nor any articles have to create the programm but I did read some tutorials on xml.