Multi-robot patrol simulation

This package is modified from [patrolling\_sim](https://github.com/davidbsp/patrolling_sim).

The main differences are as follows:

1. [LCM](https://github.com/lcm-proj/lcm), a library independent of ROS, is used to exchange messages between robots in both simulations and real robot experiments.

2. A GUI monitoring program is implemented to visualize the position of robots and control the start as well as stop of the simulation.

3. The expected reactive (ER) algorithm for multi-robot patrol is added in this package.

Test environment

Ubuntu 12.04

Boost 1.46 (Debian’s default Boost version, installed with Ubuntu 12.04 already)

[ROS Hydro](http://wiki.ros.org/hydro/Installation) and package: [stage\_ros](http://wiki.ros.org/stage_ros)

[LCM v1.0.0](https://github.com/lcm-proj/lcm) (Lightweight Communications and Marshalling)

Python 2.7 and package: [PIL(Python Imaging Library)](http://pythonware.com/products/pil/) or [pillow(The friendly PIL fork)](https://python-pillow.org/)

Quick Start

1. [Create a ROS Workspace](http://wiki.ros.org/ROS/Tutorials/InstallingandConfiguringROSEnvironment) : “~/catkin\_ws/src”

2. Move the source folder to “~/catkin\_ws/src” and rename it as “my\_patrol\_sim”

3. Build the workspace:

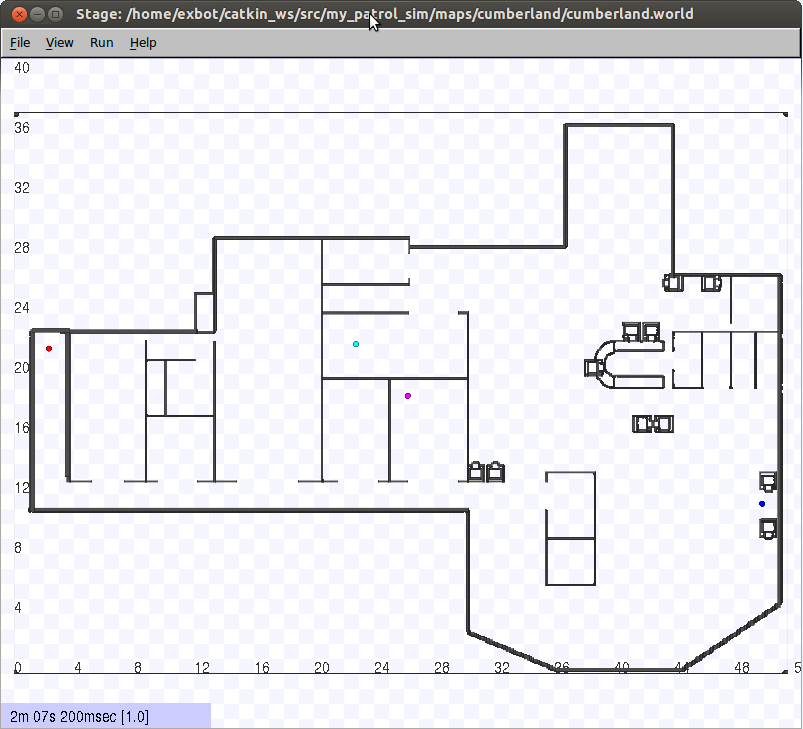
$ cd ~/catkin\_ws/

$ catkin\_make

4. Open a terminal and run:

$ source ~/catkin\_ws/devel/setup.bash

$ roslaunch my\_patrol\_sim navigation\_multi\_robot.launch



(Bringup the Stage simulator and ROS navigation module)

5. Open a new terminal and run:

$ source ~/catkin\_ws/devel/setup.bash

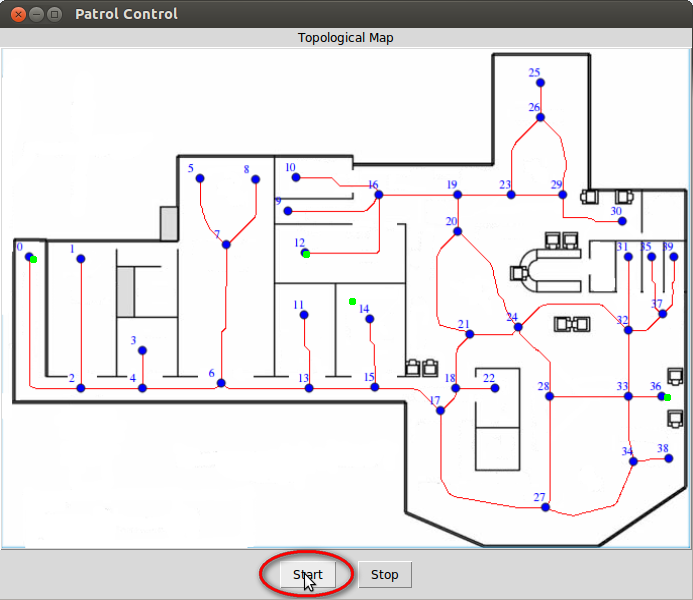
$ roslaunch my\_patrol\_sim patrol\_multi\_robot.launch

(Start the patrol decision program)

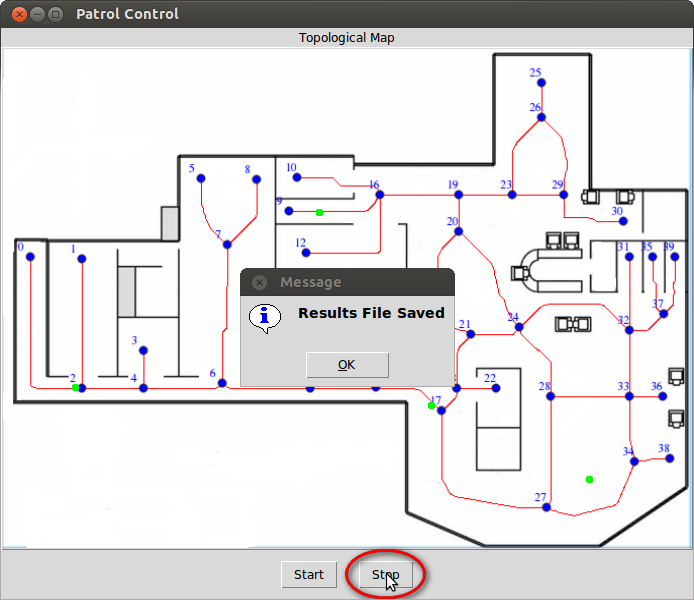
6. Open a new terminal and run:

$ cd ~/catkin\_ws/src/my\_patrol\_sim/scripts/my\_monitor

$ python my\_monitor.py cumberland 4

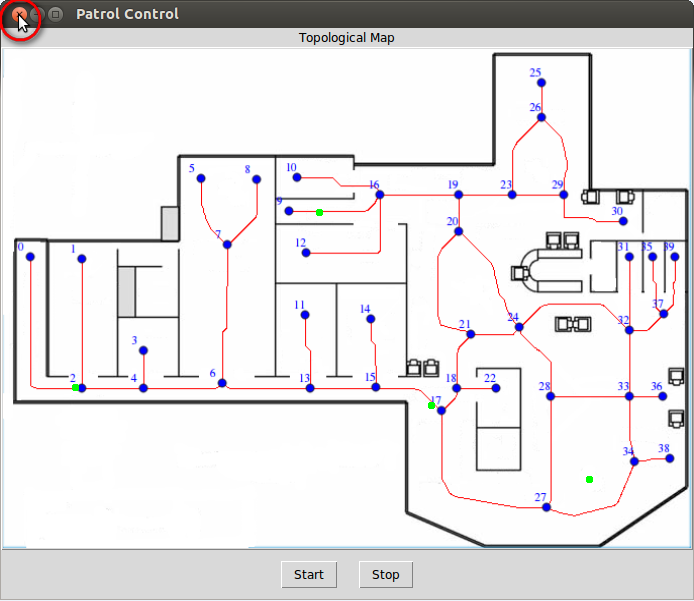


Click the button “Start” to start the simulation.



Click the button “Stop” to stop the simulation.

The statistical result will be saved in the folder “scripts/my\_monitor/results”, named as “cumberland\_4\_results.txt”



Click the Close button in the upper left corner to exit the monitor program.

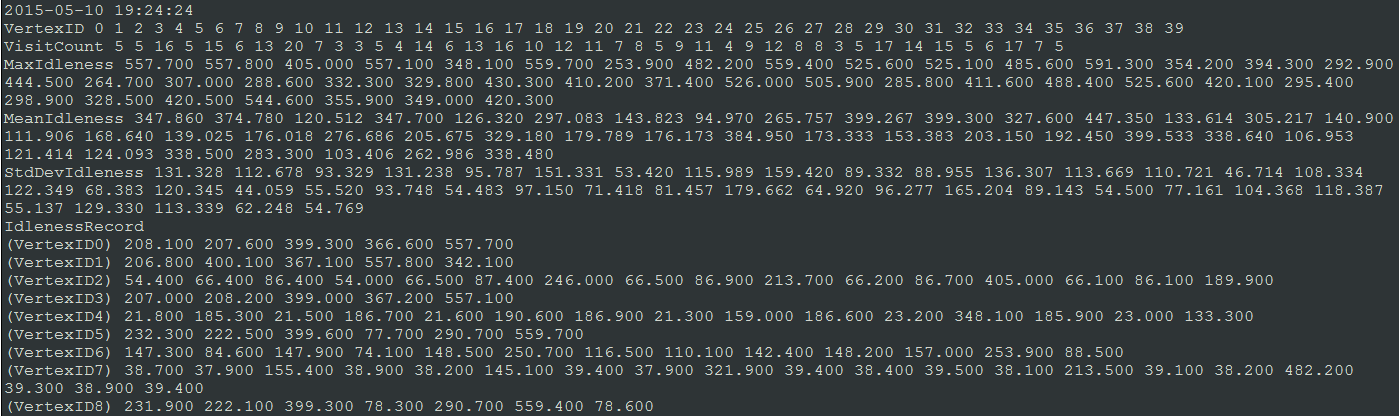
(Use “Ctrl-C” in the terminal will not make the program exit completely.)

7. To restart the simulation, you need to stop all the programs above and redo step 4 to 6.

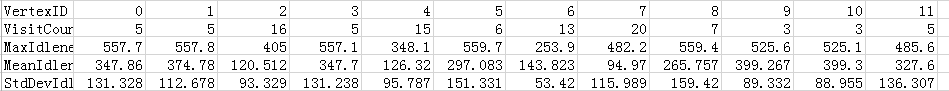
8. The statistical result

In the file “cumberland\_4\_results.txt”, the content is similar to the picture below. The term “VertexID” makes a list of vertex ID in graph cumberland (40 vertices). The term “VisitCount” means the number of visits to the vertex; “MaxIdleness” means the maximum idleness of the vertex during patrolling; “MeanIdleness” means the average idleness and “StdDevIdleness” means the standard deviation. The term “IdlenessRecord” gives the record of each vertex, and the number of items in a row equals to the “VisitCount” of the corresponding vertex.

More information can be found in the source file “scripts/my\_monitor/my\_monitor.py”.

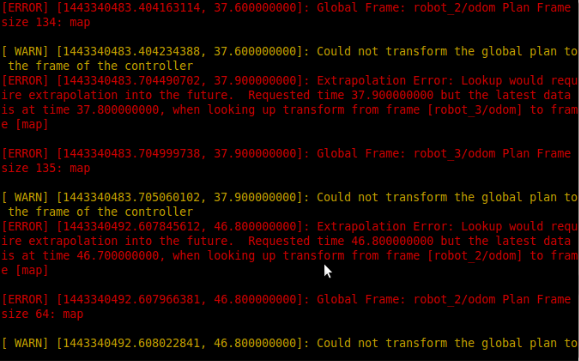


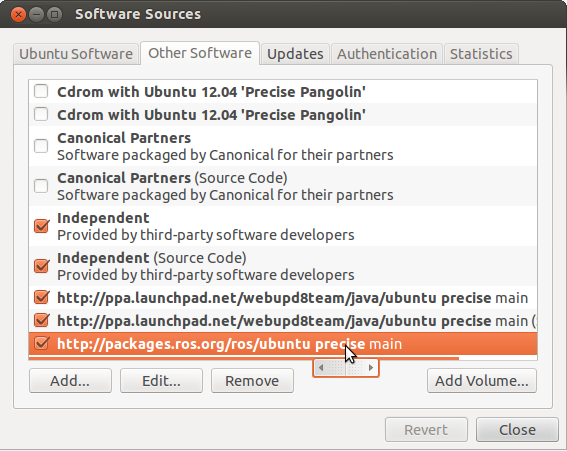
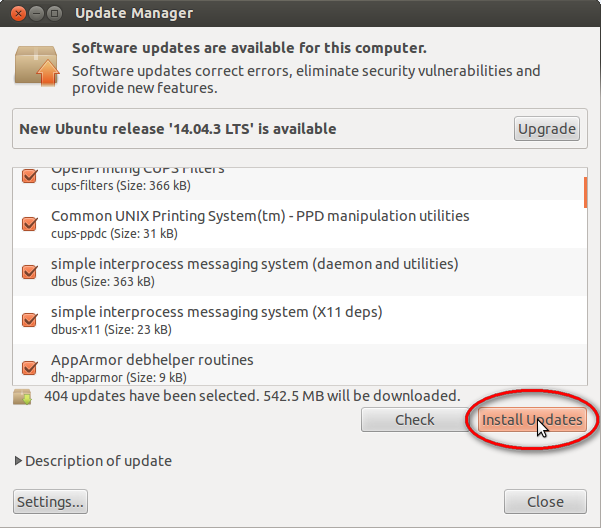
You can import the data into excel like the picture below, or use other methods to process it.

…

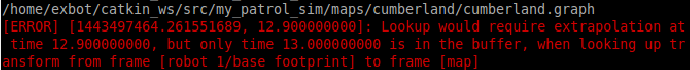
Possible problems

1. If you meet an error like this, it may be the problem related to [tf package in ROS](http://wiki.ros.org/tf). You can install updates to fix it (from <http://packages.ros.org/ros/ubuntu> precise main).

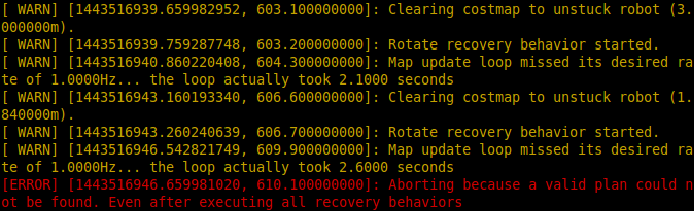


2. An error like this may occur when you run “roslaunch my\_patrol\_sim patrol\_multi\_robot.launch”. It is caused by ROS tf transform, and does not affect the program.



3. An error like this may occur in the terminal which runs “roslaunch my\_patrol\_sim navigation\_multi\_robot.launch”. It is caused by ROS navigation module, and means that robots are likely to be stuck with each other.



4. The scripts “initialize\_sim.py”, “my\_monitor.py” and “plan\_center.py” rely on their relative path, so you need to change directory to the folder that contains them before use.

Usage

**1. Simulation with different maps and different numbers of robots**

The files named “\*.world” in folder “maps/cumberland” and “maps/grid” contain the configuration for the map and number of robots for the Stage Simulator.

The file “launch/navigation\_multi\_robot.launch” contains the navigation configuration according to the number of robots.

The file “launch/patrol\_multi\_robot.launch” contains the patrolling configuration according to the number of robots.

You can configure these three files to run simulation in any map. You can also use the script “scripts/initialize\_sim.py” to configure some specific parameters quickly as follow:

$ cd ~/catkin\_ws/src/my\_patrol\_sim/scripts

$ python initialize\_sim.py <map\_name> <robot\_number>

The <map\_name> can be “cumberland” or ”grid”. The <robot\_number> can be “1”, ”2”, ”4”, ”6”, “8” or “12”. For example:

$ python initialize\_sim.py cumberland 8

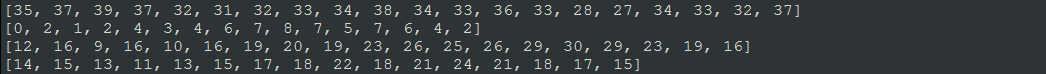
The script can generate the files “\*.world” and “\*.launch” according to the input information. More information can be found in the source file.

**2. Simulation with different patrol algorithms**

The algorithms are stored in source files “src/my\_patrol\_robot/my\_algorithms.cpp” and “src/my\_patrol\_robot/algorithms.cpp”. The function “TaskExecution::decide\_next\_vertex(...)” in the file “src/my\_patrol\_robot/task\_execution.cpp” calls one of these algorithms. You can edit this function to choose different patrol algorithms.

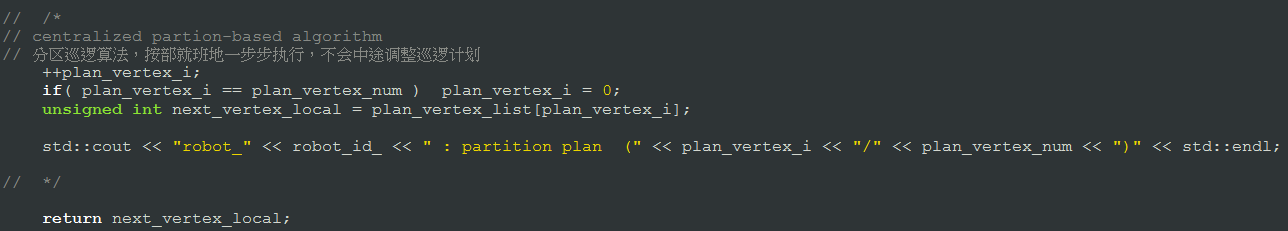
**3. Simulation with the centralized partition-based algorithm**

The usage of centralized algorithm is different from distributed algorithms above. The plans of all robots are calculated off-line, and stores in the folder “scripts/my\_monitor/plan\_files”. The contents of plan files are shown below (“cumberland/plan\_cumberland\_4.txt” for example):



Each row corresponds to the path of each robot (4 robots: robot\_0 ~ robot\_3), which means the robot need to visit these vertices sequentially and circularly.

You need to edit the function “TaskExecution::decide\_next\_vertex(...)” in the file “src/my\_patrol\_robot/task\_execution.cpp” to choose the centralized partition-based algorithm, as shown below:



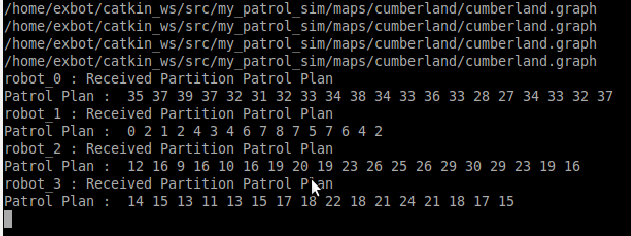
Then build the package and run the program as shown in Quick Start. Before click the button “Start” in the monitor, open a new terminal and run:

$ cd ~/catkin\_ws/src/my\_patrol\_sim/scripts/my\_monitor

$ python plan\_center.py cumberland 4

($ python plan\_center.py <map\_name> <robot\_number>)

This script reads the plan file and send the plan to the patrol decision program of each robot. The output of patrol decision program is shown below:



Finally, you can start the simulation and record the result.

**4. Simulation with special configurations**

If you want to change the velocity of the robot, you can refer to files “launch/hete\_\*.launch” to override some parameters, or directly edit your launch files.

If you want to test the impact of unreliable communications, you can edit the file “src/my\_patrol\_robot/message\_listener.h”.

The program would ignore the received message with a certain probability, so as to simulate the error rate of communication. The variable “int Handler::com\_rate\_” which represents the percentage of reliability can be set from 0 to 100, and 100 means to ignore no message.

The program would only receive messages from robots whose distance to the robot is less than “int Handler::com\_dist\_” meter, which represents the restriction on communication range. You can also set it to a large number, so as to relax this restriction.

The program would delay for a certain duration after receiving messages, and then update information used in decision, so as to represent the delay in communication. The delay time can be set in the variable “int Handler::com\_delay\_”. “com\_delay\_=0” means there is no delay in communication, and “com\_delay\_=10” means the delay time is 10 seconds.

Structure

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| my\_patrol\_sim | data\_type\_lcm | exlcm | header files for C++ to use LCM messages | |
| \*.lcm | [LCM messages’ definition](http://lcm-proj.github.io/tut_lcmgen.html) | |
| launch | includes | the included files for \*.launch | |
| \*.launch | roslaunch files used to start simulations | |
| maps | cumberland | files for the map cumberland | |
| grid | files for the map grid | |
| initial\_poses.txt | file that stores initial positions of robots | |
| param\_sim | \*.yaml | configurations for navigation | |
| scripts | my\_monitor | exlcm | files for Python to use LCM messages |
| maps | the same as “my\_patrol\_sim/maps” folder  used by “my\_monitor.py” |
| plan\_files | files for the partition plan  used by “plan\_center.py” |
| results | results of simulations  \*\_results.txt : average idleness etc.  \*\_robotpath.txt : paths of robots  \*\_timeidleness.txt : instantaneous idleness along time |
| my\_monitor.py | the monitor program |
| plan\_center.py | program used for the partition plan |
| initialize\_sim.py | python script to configure simulations | |
| src | my\_patrol\_robot | getgraph.h/cpp | functions to read the \*.graph file |
| algorithms.h/cpp | functions of different patrol algorithms |
| my\_algorithms.h/cpp |
| global\_variables.h/cpp | global variables for multi thread |
| message\_listener.h/cpp | thread that listens to LCM messages |
| task\_execution.h/cpp | thread that executes the patrol task |
| my\_patrol\_robot.cpp | the main thread |
| CMakeLists.txt | configurations for ROS catkin make | | |
| package.xml |

Summary

The programming style of this package may be not good. The selection of different algorithms need to modify the source code and rebuild, rather than pass parameters easily. It is grateful for the effort to improve this package. Any questions, please email ycb11 (at) mails.tsinghua.edu.cn .