How to program micromouse in ROS - EN

Goal of this tutorial is to show how to control a robot simulated in Gazebo with prepared ROS packages.
It will be described how to run simulation, configure maze and how to create control algorithm in C++ and Python.

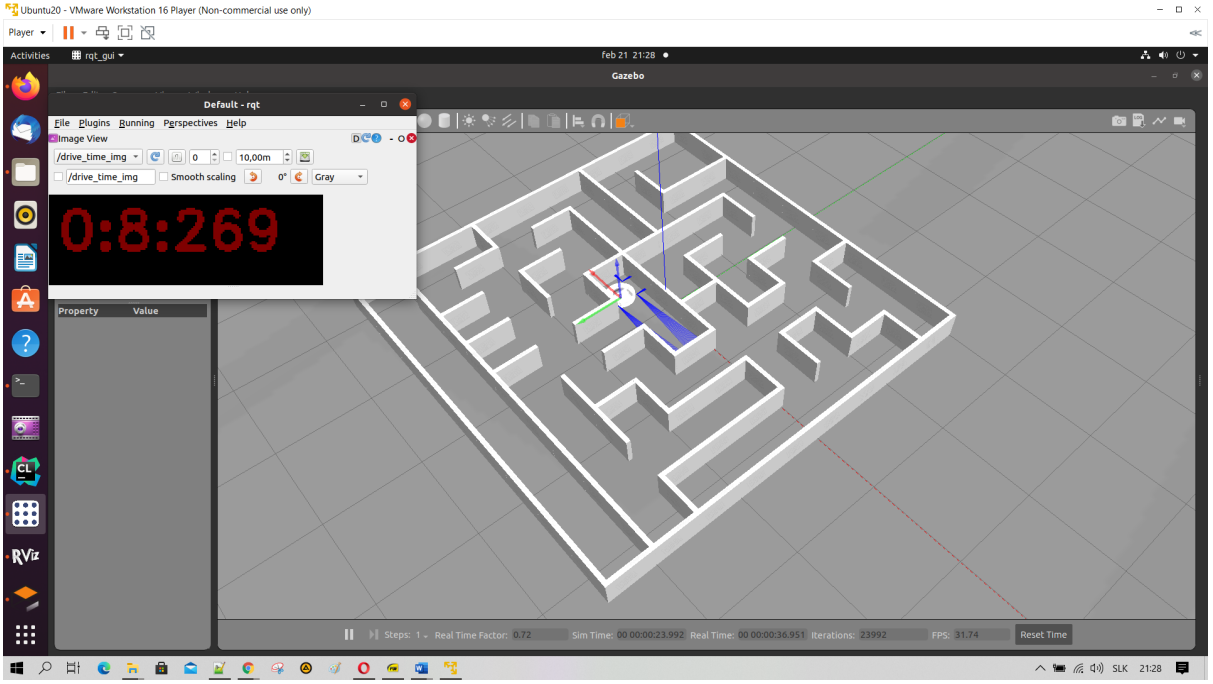
Running the code

As a prerequisite is installed operating system Ubuntu 20 and ROS Noetic full desktop. After that it is possible in the created and initialized catkin_ws to src directory to clone or to download packages from the repository:
https://github.com/Smadas/irobot_create_ros - model iRobot Create
https://github.com/Smadas/maze_generator_ros - generovanie modelu bludiska z obrázka
https://github.com/Smadas/create_control_ros - riadiaci algoritmus pre simulovaný robot
Next, it is necessary to compile the downloaded source codes by running a console command catkin_make from directory catkin_ws.

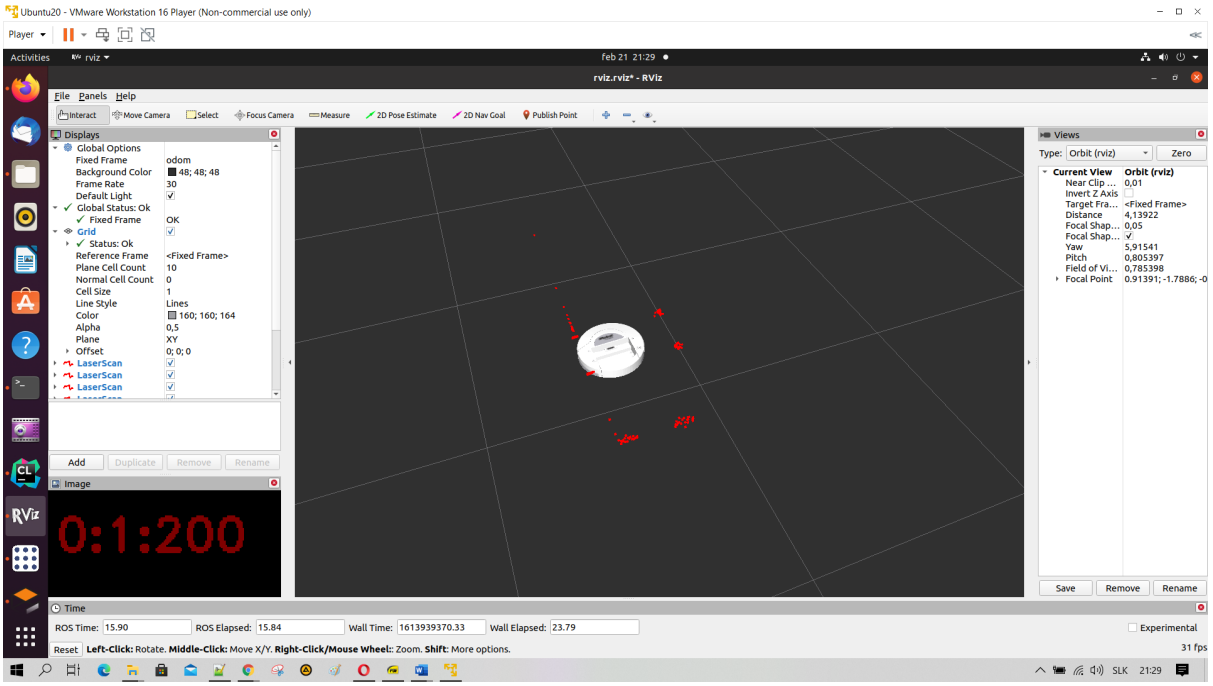
After that it is possible to run one of the launch files from bash console depending on which we want to run simulation, control or both at once and whether we want c++ control or python control:

- control.launch
- sim_control.launch
- sim.launch
- control.launch
- sim_control.launch
- sim.launch

We can run the launch file with appropriate console command:
roslaunch create_control <launch_file_name>
Then we can end all launched processes by pressing ctrl+c in the console in which we have run the previous command.



Gazebo simulation with maze, robot and timer.



Rviz visualization with robot, rangefinders and timer.
In the event that the sidebar malfunctions, to get it back to work you just have to resize it.

What will be running

1. Maze generator, creates urdf and sdf model of a maze from png picture. Further, it computes coordinates of starting and goal area and starting point of the robot.
2. Parameters are put into the parameter server.
3. Gazebo simulation with generated maze and iRobot Create on coordinates which were previously computed.
4. Robot control algorithm (c++ or python).

5. Rviz - visualization of the robot with all sensors and timer.
6. Timer - independent GUI displaying time elapsed since the robot left starting area with its center of gravity until it crosses into the goal area (if only an empty window is show a choose correct plugin from menu>plugin>visualization>Image view and type into textbox /drive_time_img). The final elapsed time is saved into create_control/results/results.txt always after the robot has crossed into the goal area.

The results.txt file will be overwritten every time the simulation is newly run.
Warning: After each change of the maze it is necessary to run and shutdown the whole simulation to allow for the changes to be present.

Description of the simulated robot

Simulated robot is model of real iRobot Create robot with differential chassis, which parameters are:

- wheel separation 0.26m
- wheel diameter 0.066m

Simulated robot has three types of sensors:

- IMU - determines rotation around x, y, z axes (radians).
- N-coders - determines position of wheels in radians.
- Rangefinders determining distance from obstacles
    - angle of projection 0.2rad
    - distance range 0.02m to 1m
    - gaussian distribution error 0.01m

You can see parameters in detail in mode-1_4.urdf and sdf files.

Generating new maze

By running the launch file which executes simulation, also a program for generating a maze is executed. Maze is generated from a png picture located in maze/maze_desc/ in which are maze descriptions.
New descriptions can be created by simple bitmap editors like pikoPixel, as an example you can use files that are already present in maze_desc.
After the creation of description it is necessary to adjust configuration file maze/config/maze_config.yaml, where are stored paremeters for walls, start, goal and name of the description file.
Finally generator_maze creates a new model of a maze and a configuration file with coordinates of the starting and goal area and coordinates o robot starting point. These coordinates are taken into consideration by the timer.

Programming of the control algorithm

It is possible to choose from c++ (create_control/src/simple_control.cpp) and python (create_control/scripts/simple_control_py.py) to create the control. The files contain sources with appropriate comments for understanding of even beginners. For ease of use are conveniently placed comments where your code can be placed //* description*// for c++ and

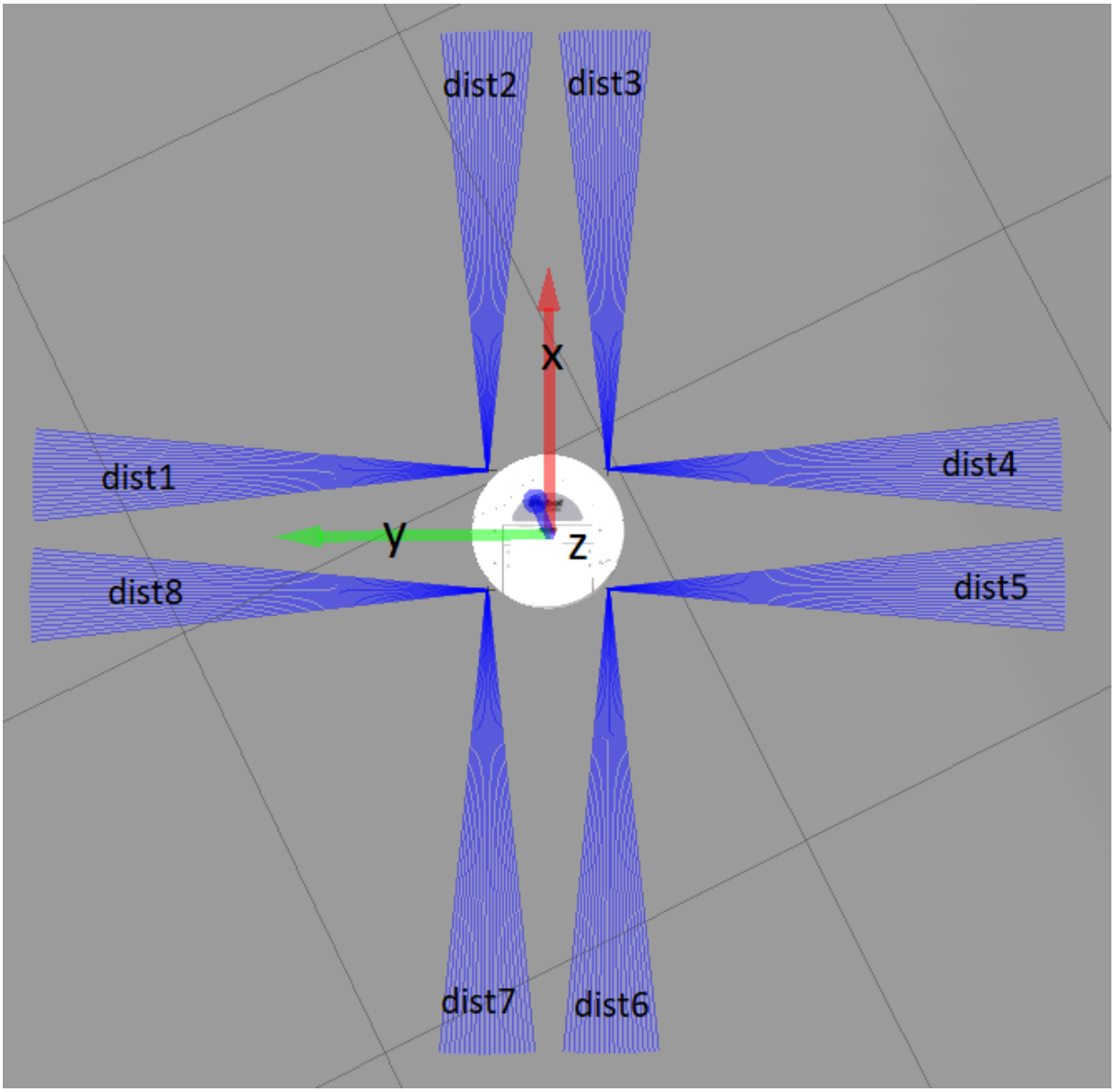
##* description *## for python. Obviously, it is possible also to add more files and edit the existing ones.
Before running of the python code it is necessary to set the python script as executable by running the following command in its directory:
chmod +x create_control_py.py
Do not forget after each change to the source code to run compilation by running catkin_make command in catkin_ws directory.
Robot is controlled with an infinite loop which regularly reads robot sensors and writes velocities into the robot.

Control message

Robot is controlled with a message of a type Twist, which consists of rotational velocities around x, y, z axes and translational velocity in x, y, z axes. Robot then travels with the resulting speed. Robot is controled with topic cmd_vel with member parameters:
geometry_msgs::Twist cmd_vel
cmd_vel.angular.x = 0
cmd_vel.angular.y = 0
cmd_vel.angular.z = 0
cmd_vel.linear.x = 0
cmd_vel.linear.y = 0
cmd_vel.linear.z = 0

Data from sensors

Robot publishes values from sensors on topic sensors with defined message and member parameters:
create_control::create_sensors sensors
sensors.dist1 = 0 - (meters) average distance of obstacle from left front left laser scanner (simulated ultrasound sensor)
sensors.dist2 = 0 - all consecutive sensor in clockwise direction
sensors.dist3 = 0
sensors.dist4 = 0
sensors.dist5 = 0
sensors.dist6= 0
sensors.dist7 = 0
sensors.dist8 = 0
sensors.imu_p = 0 - angle around x axis (radian)
sensors.imu_r = 0 - angle around y axis
sensors.imu_y = 0 - angle around z axis
sensors.left_wheel = 0 - angle of left wheel (radian)
sensors.right_wheel = 0 - angle of right wheel (radian)



For advanced developers

Examples of control algorithms are created in the most simple way possible to allow even beginners to use the packages, but it is possible to use all other available tools, like you can change te c++ and python code to classes.
Further it is possible to use other libraries, modules and ros packages. In this case it is necessary to edit CmakeLists.txt and package.xml in the appropriate package:

- into CmakeLists.txt find_package(....) add name of the package you are adding
- into CmakeLists.txt include_directories(.....) add a path to the header files
- into package.xml dependencies to the package you are adding (use in similar way is in the example package).

Further it is possible to to create own ROS topics or services for receiving and sending data. How to do this you can find in: http://wiki.ros.org/ROS/Tutorials