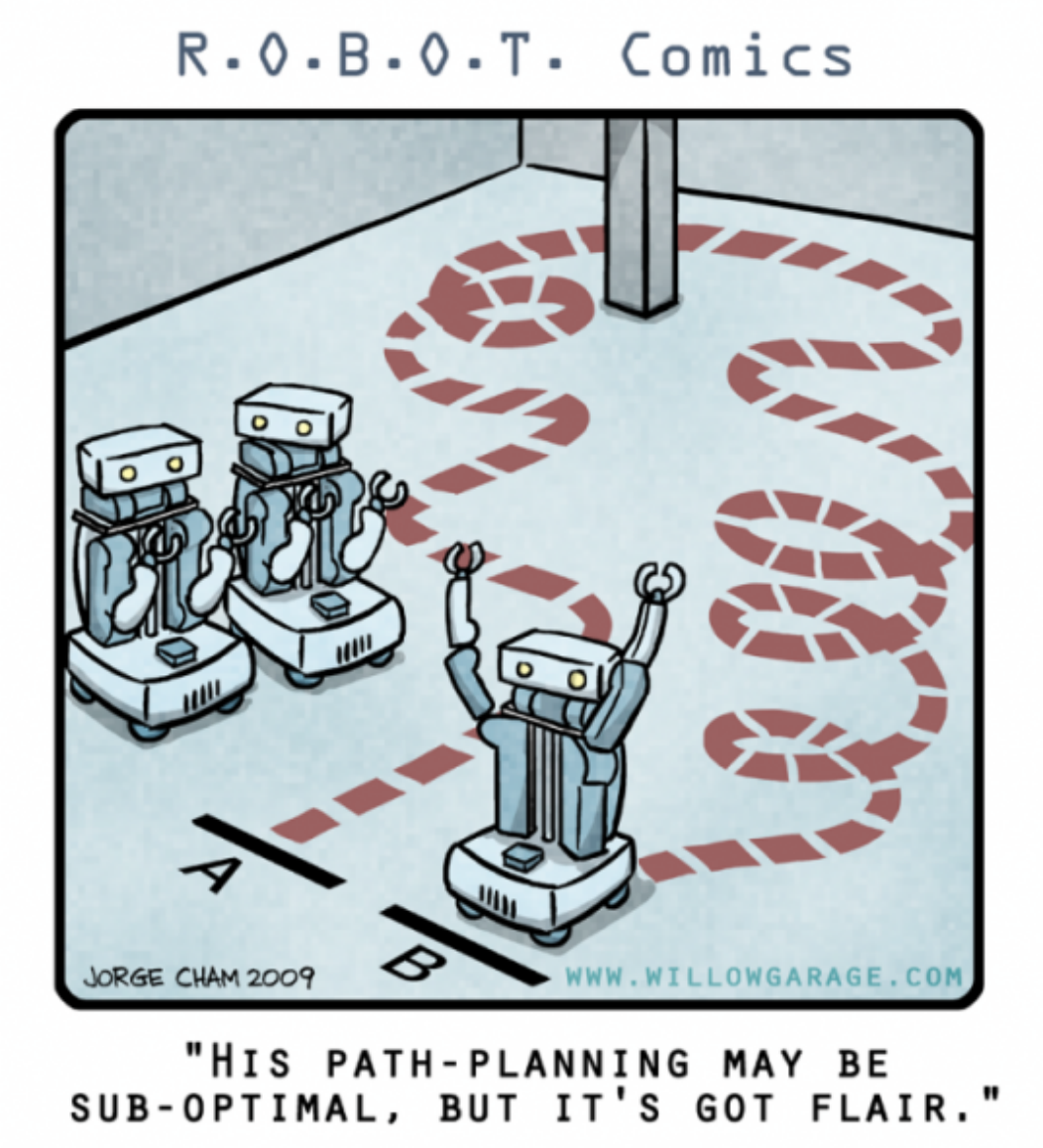
Advanced Robotics

Project Report

Active SLAM



XU LIAN
11547229

Introduction

The primary difference of active SLAM or SPLAM (simultaneously planning, localization and mapping) and normal SLAM is to gain information of unknown surrounding environment autonomously. This involves additional path planning (how the robot goes to there) and control (where it should go to collect more information) mechanism.

In this project, its primary goal is to control the Fetch robot moving from one side to another in dynamic environment such as a corridor with collision avoidance in mind (for example, an emergency stop action).

Analysis

In this section, it will describe the potential tasks that should be involved in this project, and its suitable approaches.

SLAM

SLAM is the principle functionality of this project. To avoid underperformed SLAM performance in this project, using existing well designed and tested SLAM implementation for ROS is preferred.

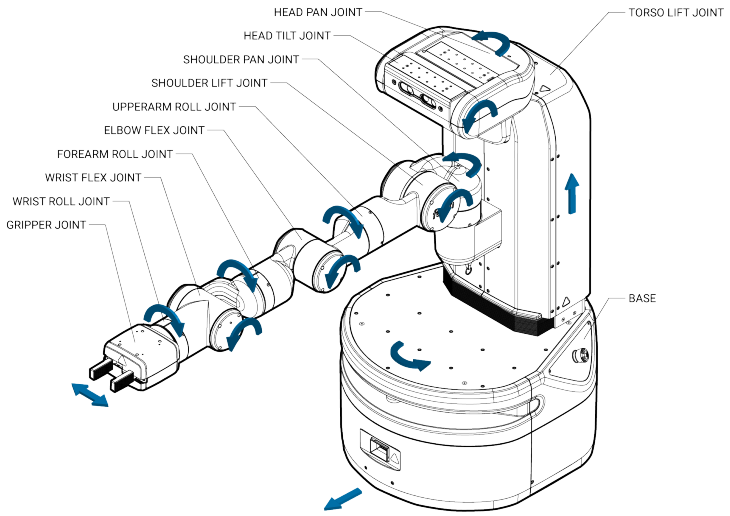


Figure 1. Fetch robot perspective

According to Fetch Robotics (2015), the Fetch robot equipped with a maximum 25 meter 220-degree laser scanner in its base and a 0.35 – 1.4-meter range RGBD sensor mounted on its head. Therefore, both 2D laser and RGBD based SLAM solutions are suitable for this specified robot.

Navigation

It is crucial for robot being able to find the most adequate pathway in known or unknown environment. Moreover, it is preferable that the navigation system could be capable to handle collision avoidance since the robot will operate in a dynamic environment.

Due to the very limited project duration, it is not viable to research and develop an entirely new navigation solution on specified platform fetch robot. Therefore, using and tuning existing navigation package for this project is the most appropriate approach.

Exploration Planning

Being able to plan the exploration autonomously is another principle feature in this project's topic Active SLAM. The preferred method is similar to SLAM and

Navigation tasks, using existing package that is suitable for specified robot running on ROS. If there is non-existing package functioning properly on specified platform, develop one based on learnt algorithms.

Project Scope

Through the project analysis, it concludes the final project scope. The overall scope of this project is to develop and deliver an implementation based on existing algorithms and methods for active SLAM with obstacle avoidance in a dynamic environment, such as large office populated with moving people and objects. It contains following objectives:

- Use and understand existing SLAM methods such as gmapping and rtabmap, to map the environment and localize the robot simultaneously by using 2D laser or RGBD sensor.

- Use and understand existing path planning and control implementations such as move_base and nav2d, to navigate the robot through given environment autonomously with reactive obstacle avoidance.

- Use or develop an implementation for exploration planning for Fetch robot running on ROS.

Implementation

In this section, it will explain the implementation produced in this project.

SLAM

Through various simulation runs, gmapping has the most impressive performance on both SLAM accuracy and software synergy which means it is working out of the box without any additional setup, tuning or work around. It is highly recommended to have software dependencies that can run as a black box without additional dependencies for projects with very limited development period such as this one.

A simple visualized SLAM result in RViz is shown in Figure 2 below.

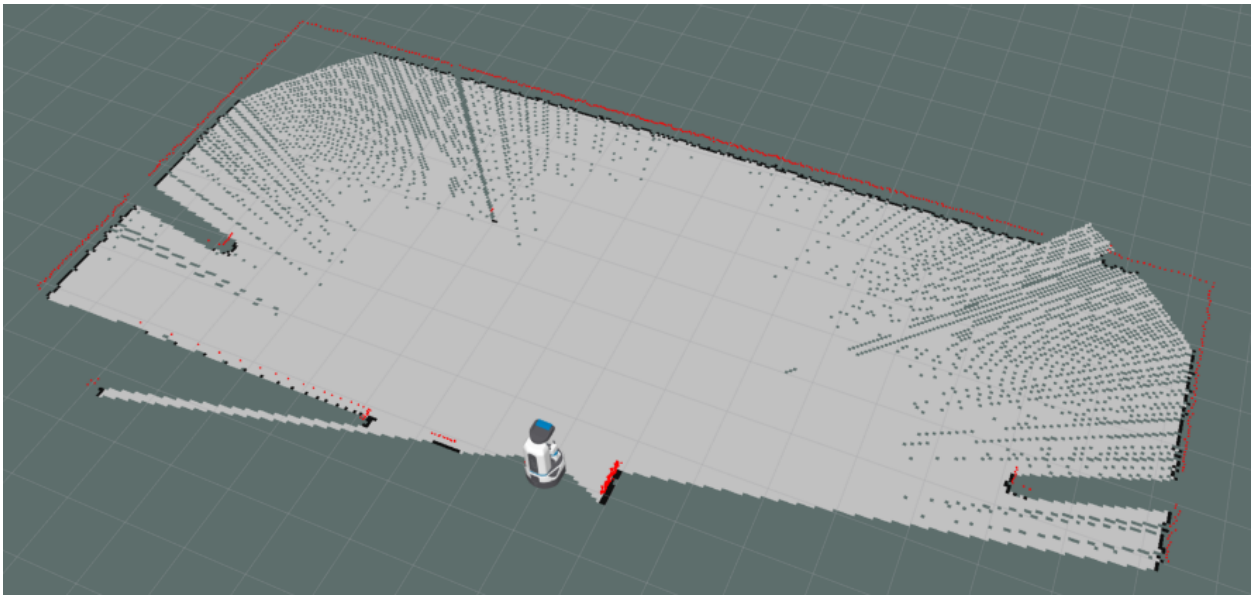


Figure 2. Gmapping visualized result

Navigation

By surfing through official navigation tutorial from Fetch Robotics, it comes clearly to that the Fetch Robotics is officially supporting move_base as their preferred navigation solution. With officially provided move_base package configuration files from the Fetch Robotics, move_base has a robust performance in the simulation on controlling the Fetch robot traveling through complex environment. The node setup diagram is shown in Figure 3 below.

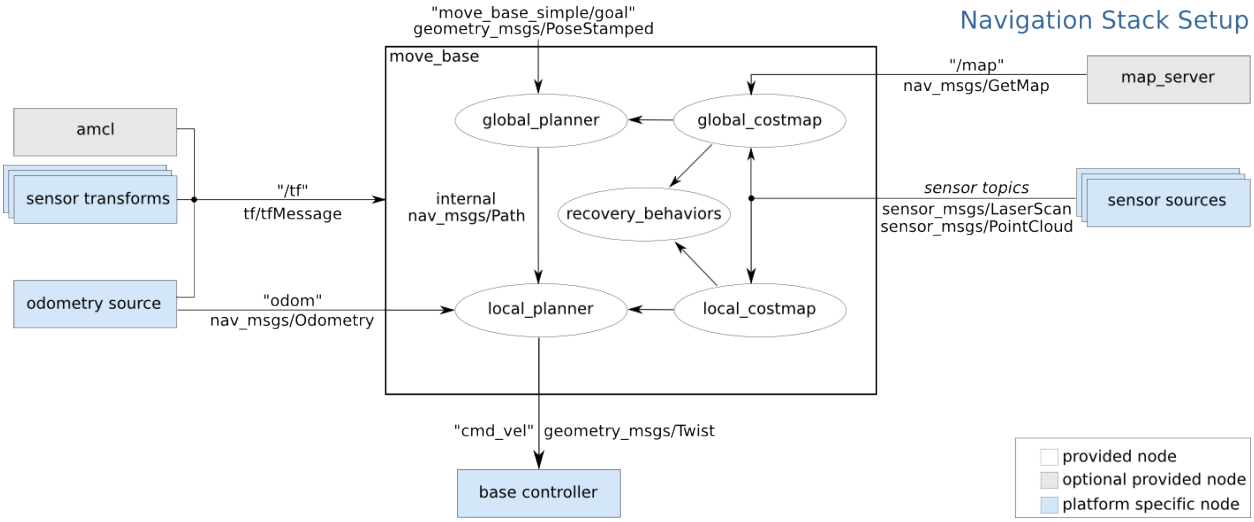


Figure 3. move_base node setup (Marder-Eppstein, E. et Al., 2016)

According to provided configuration from the Fetch Robotics (2015), move_base is being configured to use additional point cloud generated from head mounted RGBD sensor for obstacle avoidance. Its expected navigation behaviors for recovering robot from stuck state are illustrated in Figure 4 below.

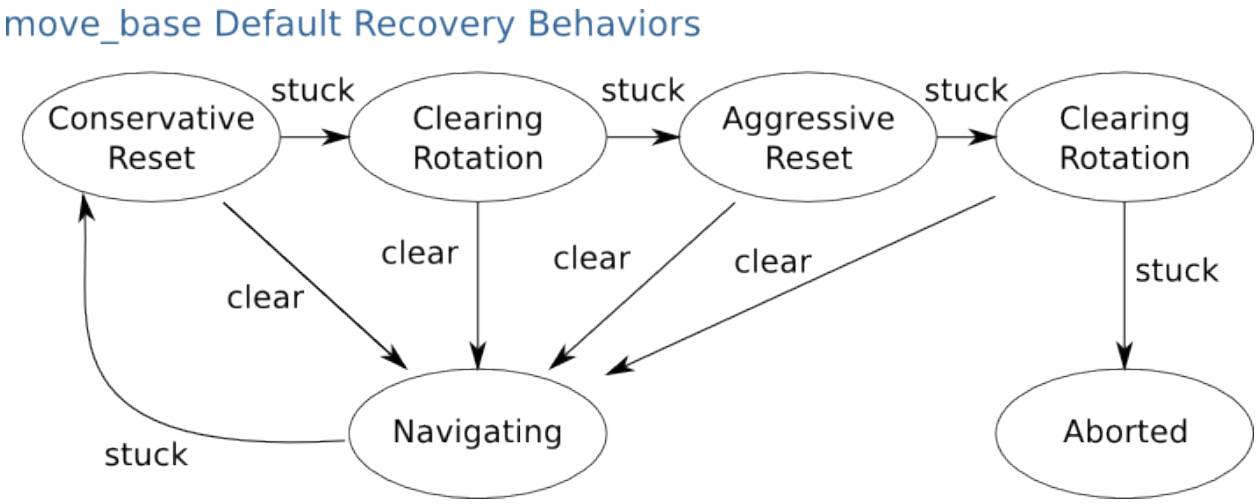


Figure 4. Expected robot behaviors (Marder-Eppstein, E. et Al., 2016)

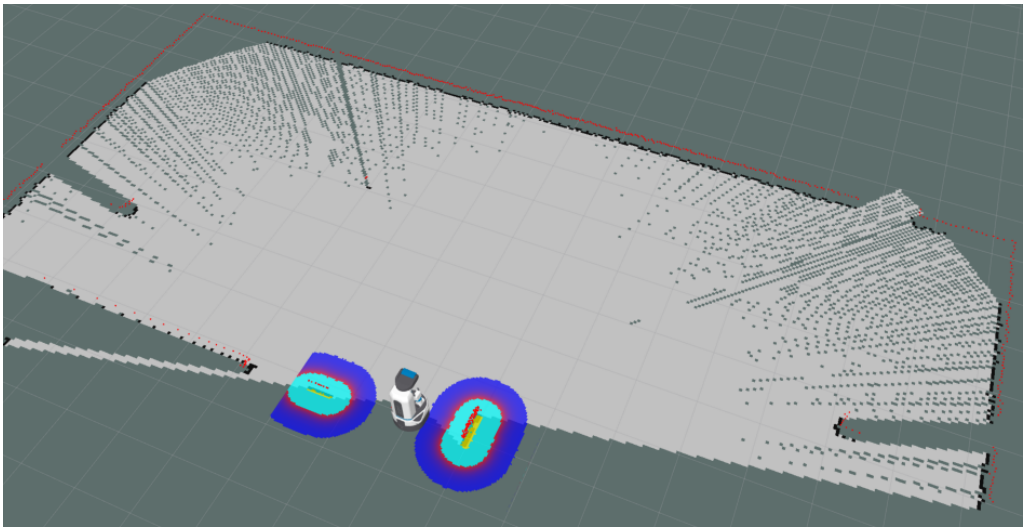


Figure 5. Local costmap

Through simulation observation and document skim, move_base will firstly generate a local costmap for pathfinding and collision avoidance, from grid map produced from SLAM node and sensor inputs such as laser scan and depth point cloud. An example result of this process is visualized in Figure 5 above. The blue area marked as the minimum safe zone that robot should keep out.

Due to the limitations of 2D laser scanning used in the SLAM solution, it is not possible to detect objects that has thin ground support structures such as table and chairs. A typical example is archived in Figure 6 and Figure 7 below marked the red dots as laser beams and white dots as the depth cloud points from RGBD sensor.

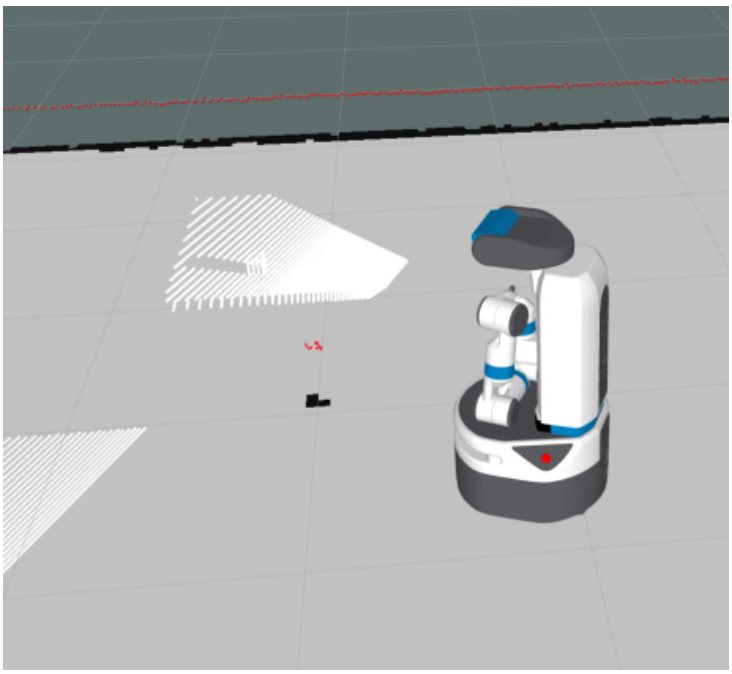


Figure 6. Situation that a table cannot be seen by 2D laser (red dots) but it is visible to RGBD sensor (white dots)

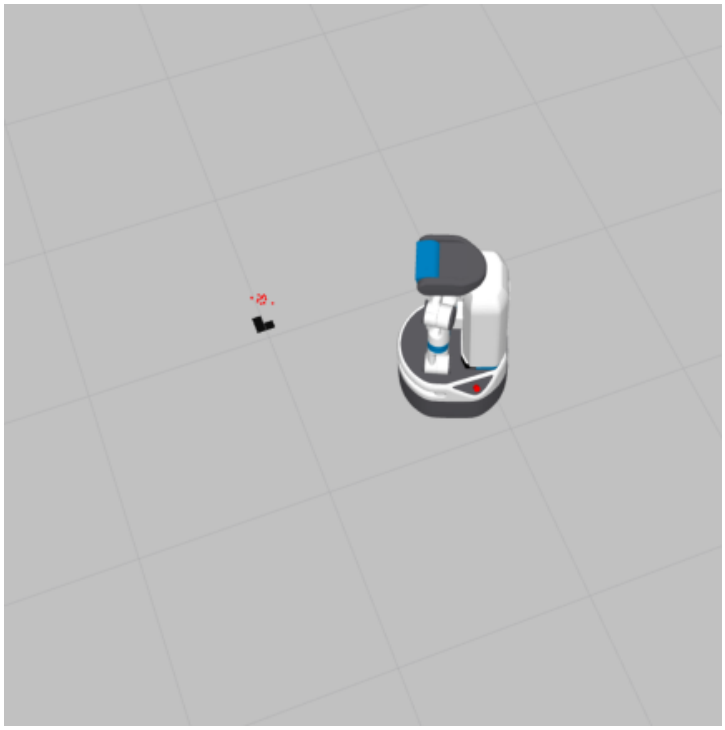


Figure 7. SLAM gird map that a table cannot be seen by 2D laser (red dots)

Fortunately, the move_base is considering the additional sensor inputs such as point cloud from RGBD camera when built local costmap for collision avoidance. The visualized costmap result for previous example used in this section is presented in Figure 8 below.

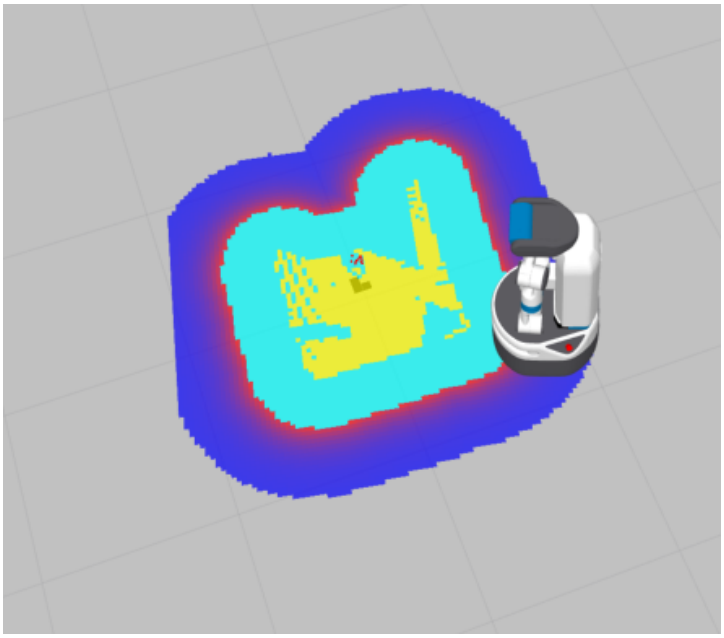


Figure 8. Local costmap using camera depth cloud to generate

Another feature provided by Fetch Robotics to move_base is the tilt_head script. It will rotate the head sensor base to face the goal of local path trajectory for gaining vital information on objects that cannot be detected by laser scanning along the navigation process. Examples of this behavior are shown in Figure 9, 10 and 11 with local path marked as dark blue and global path toward the navigation goal marked as green.

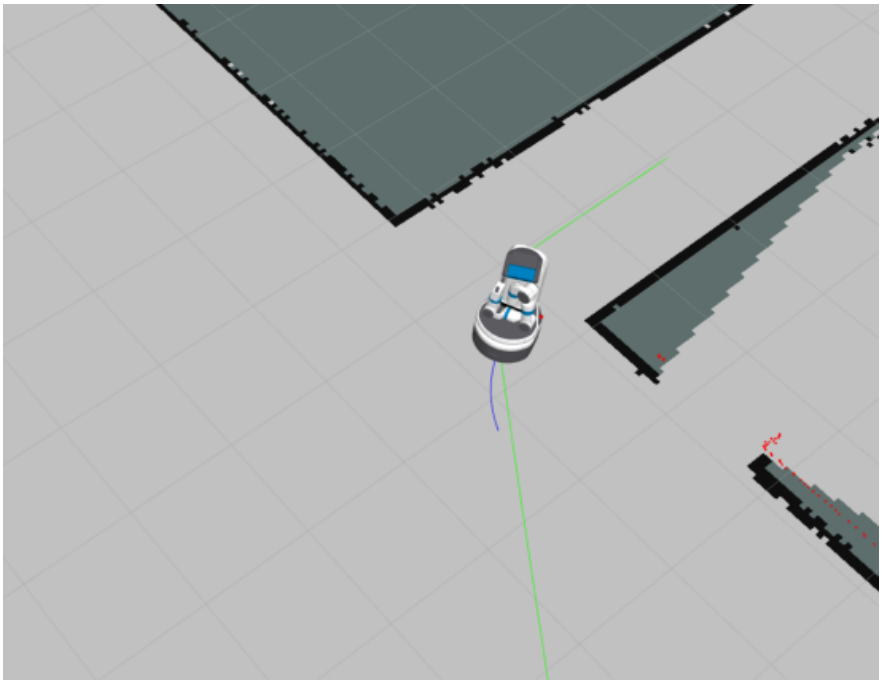


Figure 9. Global and Local path planner

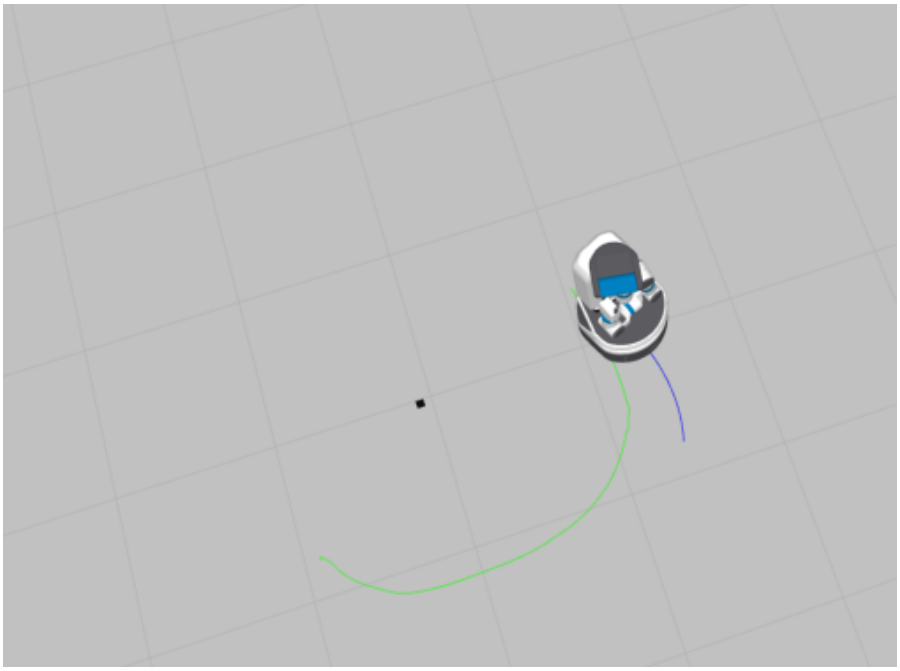


Figure 10. Head node rotation correlated to Local path planner

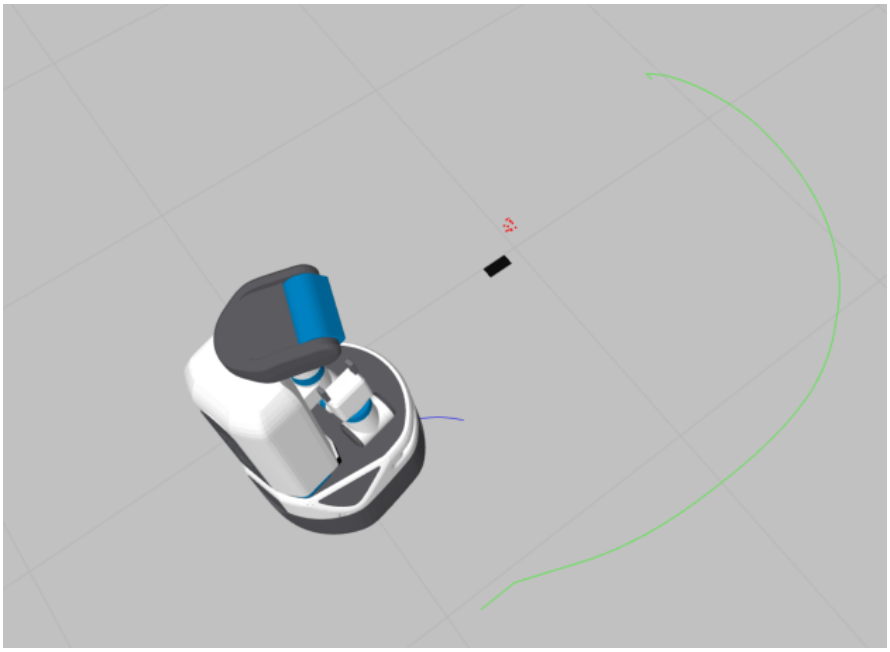


Figure 11. A closer look of head node rotation

As a result, move_base is definitely the most feasible solution on both navigation and collision avoidance.

Exploration Planner

After numerous unsuccessful attempts on using existing exploration planning packages such as nav2d Exploration, frontier_exploration and explorer from Alpen-Adria-Universität Klagenfurt, developing a simple exploration planner by using known technics is appearing to be beneficial on gaining project progression. By using obstacle inflation technics used in navigation and a slightly modified Dijkstra's algorithm generally used in pathfinding, a single robot only exploration planning node has been produced. In this section, it will focus on explain the procedures of this newly developed exploration planner.

Through the observation of navigation node move_base, it inflates not accessible cell such as walls to avoid collisions between the robot and obstacles. This inspired the first step of frontier searching that is to inflate obstacles on received occupancy grid to filter out unsafe and unreachable areas from SLAM node. The result of a typical situation is shown in Figure 12 below.

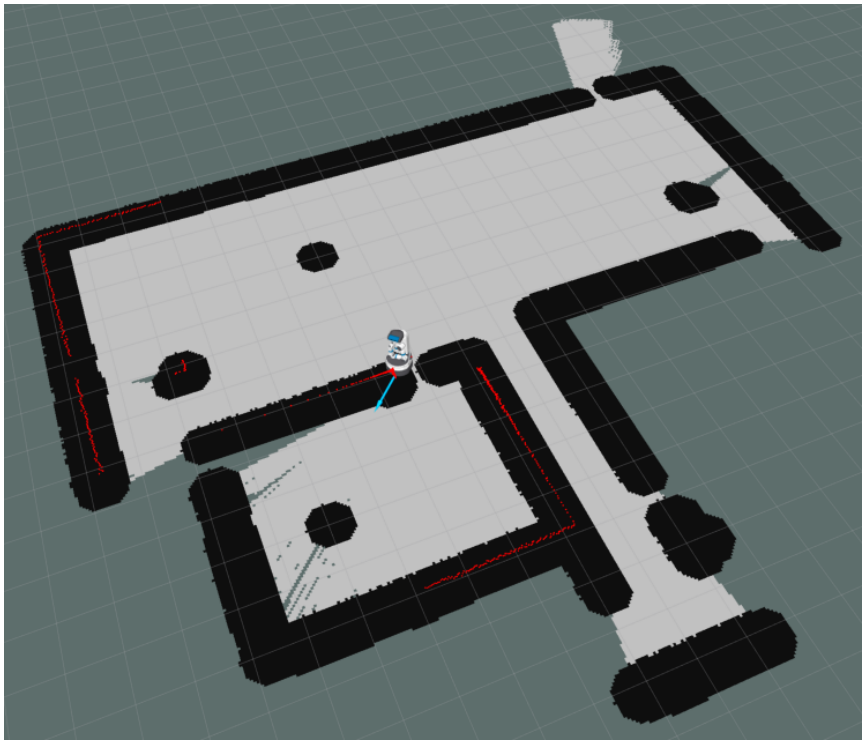


Figure 12. Step 1 inflate the received map

Once the map has been inflated, Dijkstra's algorithm will be active in use to find all frontiers (first unknown cell from known neighbors). The pseudocode of this process is presented in Table 1. In addition, an example result from previous instance is shown Figure 13 below.

Table 1. Pseudocode of frontier searching

```
Clear frontierList
Push searchOrigin into openList
While openList is not empty
    Cell = Pop First in openList
    For Movement in movementList
        If (Cell + Movement) is openGround and (Cell + Movement) is not closed
          Push (Cell + Movement) into openList
          Close (Cell + Movement)
        If (Cell + Movement) is unknowGround and (Cell + Movement) is not closed
         Push (Cell + Movement) into frontierList
         Close (Cell + Movement)
```

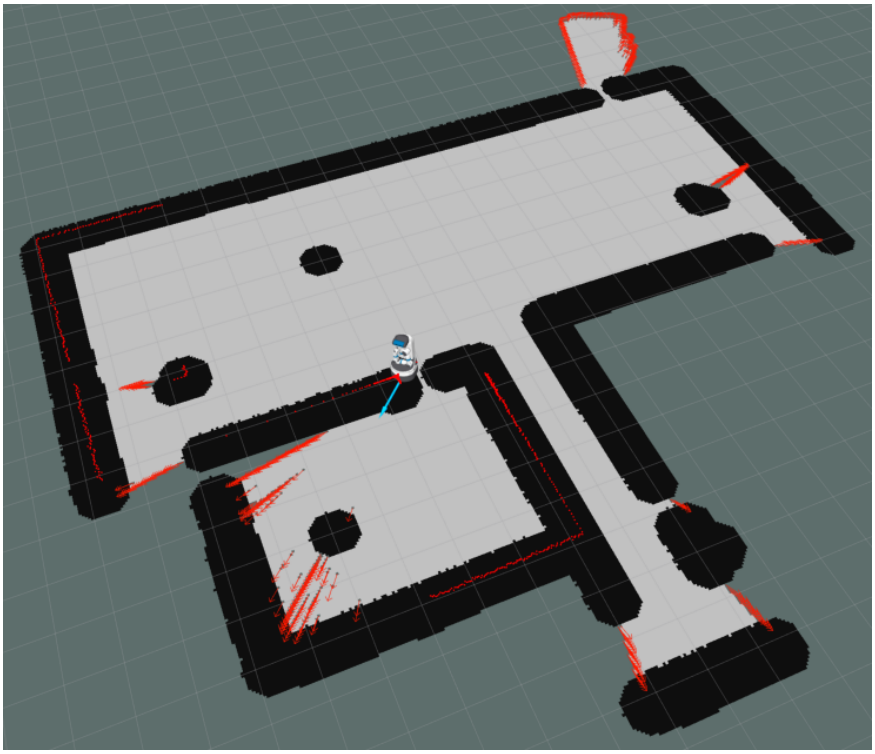


Figure 13. Step 2 find frontiers on processed map (frontier marked as red vector)

After frontiers have been determined from the grid map, the selected frontier strategy will pick out the best candidate according to predefined thresholds such as minimum and maximum distance from the robot. The pseudocode of nearest-first strategy used in this planner is presented in Table 2. In Figure 14, it is shown the selected frontier marked with green arrow from all known candidates.

Table 2. Pseudocode of nearest-first planning strategy

```
bestFrontier = frontier initialized with worst conditions
for Frontier in frontierList
  if (Frontier has lesser distance toward the robot) and (Frontier satisfies both
maximum and minimum distance toward robot)
    bestFrontier = Frontier
```

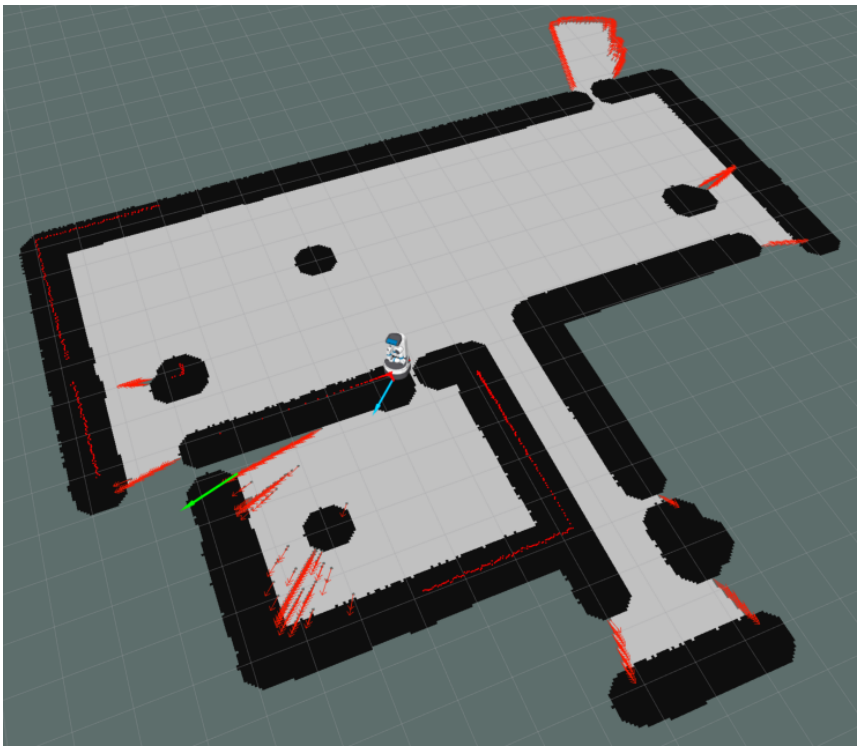


Figure 14. Step 3 choose a frontier based on the active strategy

Since the next frontier has been pinned, the exploration planner will then send the next frontier pose to navigation node. A typical virtualized result in RViz of the entire procedure is shown in Figure 15 below.

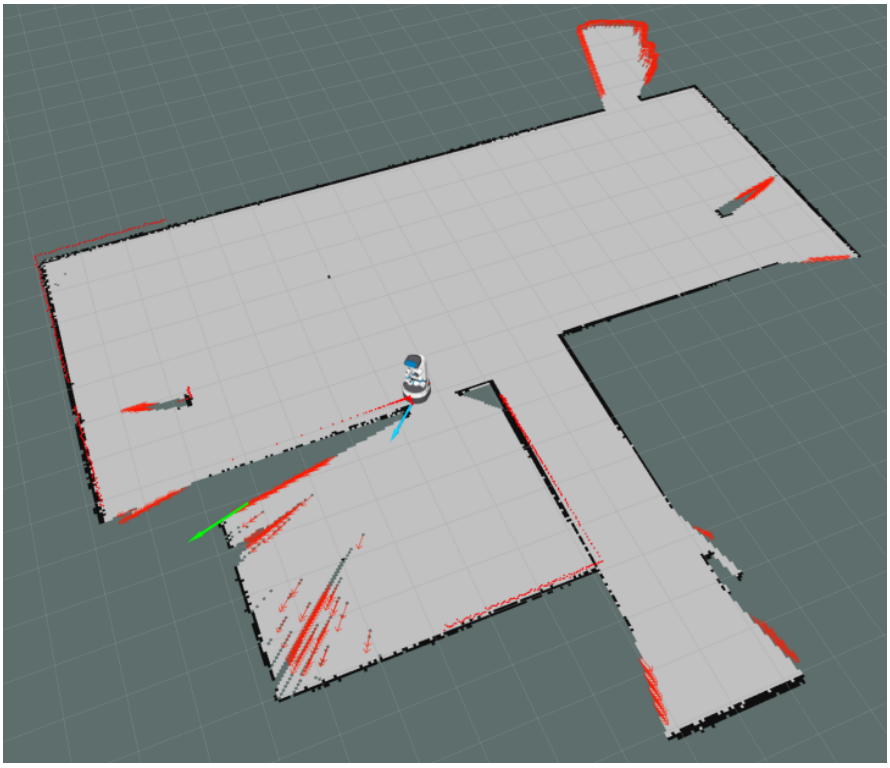


Figure 15. Final view of frontier planner

The final overall flowchart of exploration planner operating at 1 hertz frequency has been presented in Figure 16 below.

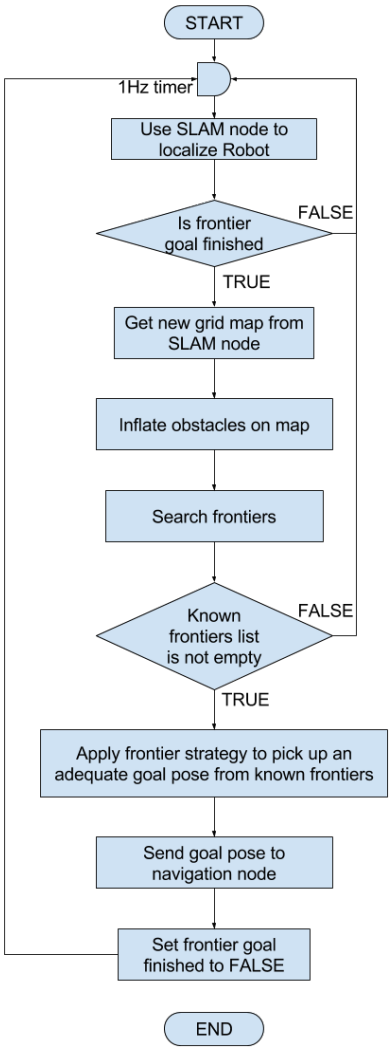


Figure 16. Overall flowchart of exploration planner

Final Result

This project has been running in the gazebo simulation numerously before the final review by supervisors. A typical result of the simulation running inside gazebo is shown in Figures below.

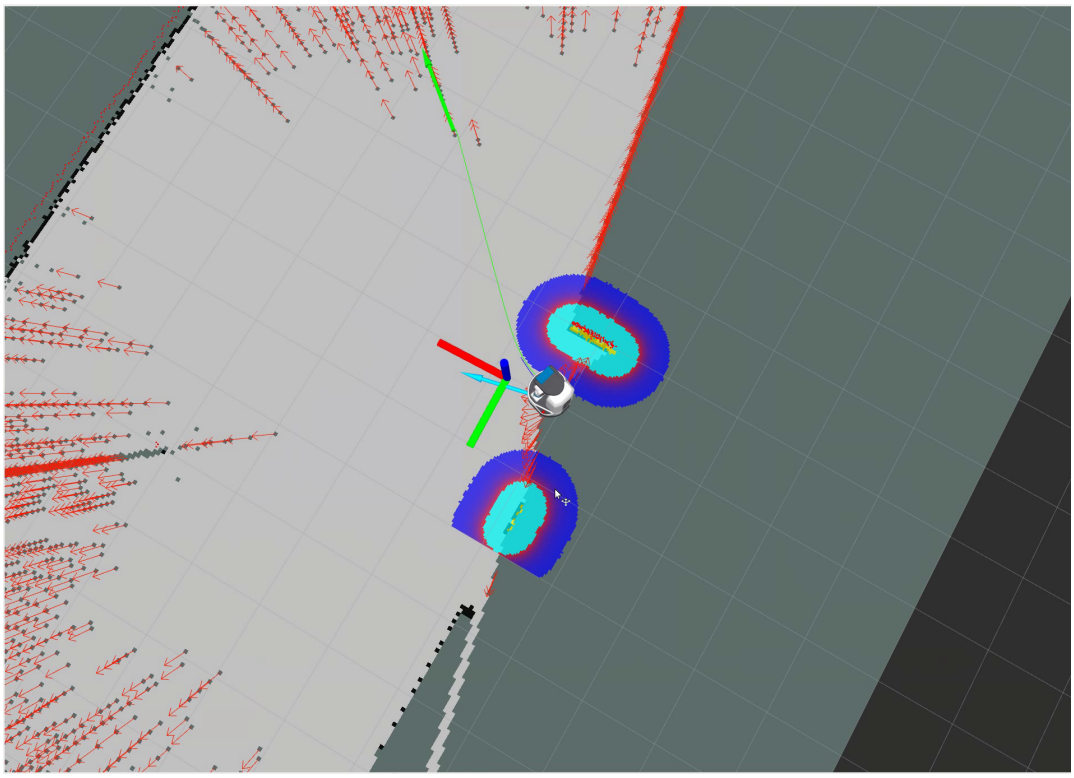


Figure 17. At the beginning, the robot moved out to the nearby uncovered grid cell

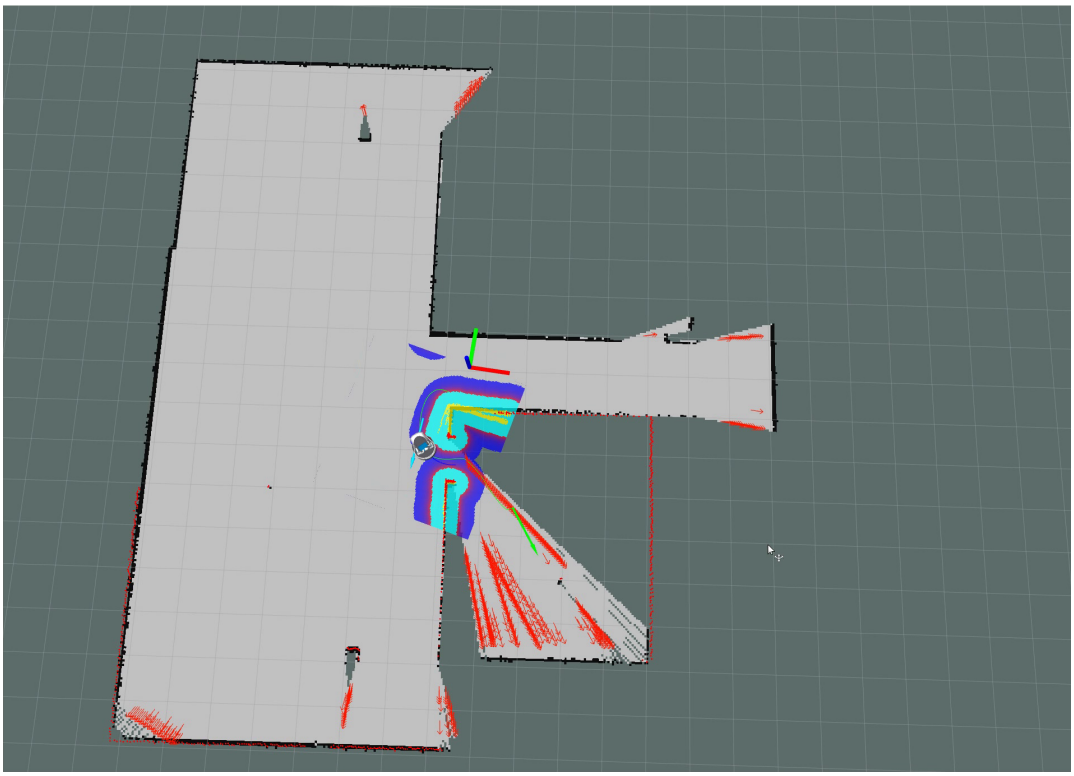


Figure 18. After the initial environment has been discovered, the robot was heading to visit the nearby room

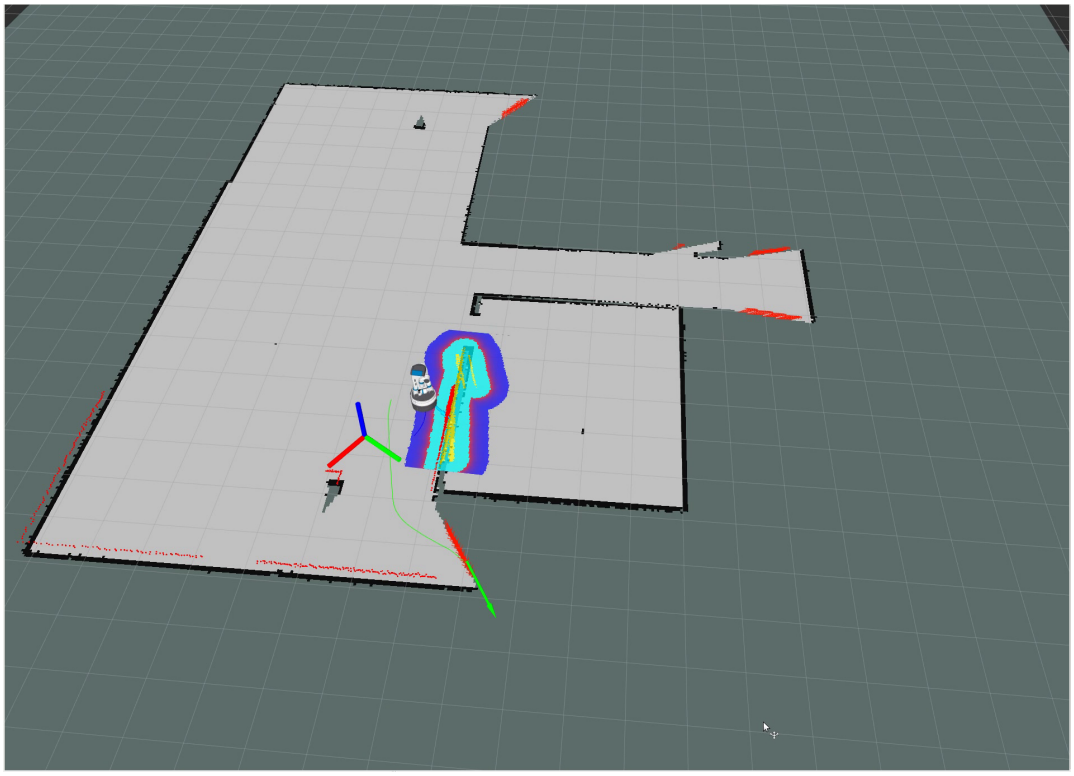


Figure 19. Once the room was uncovered, the robot was heading out to the hallway

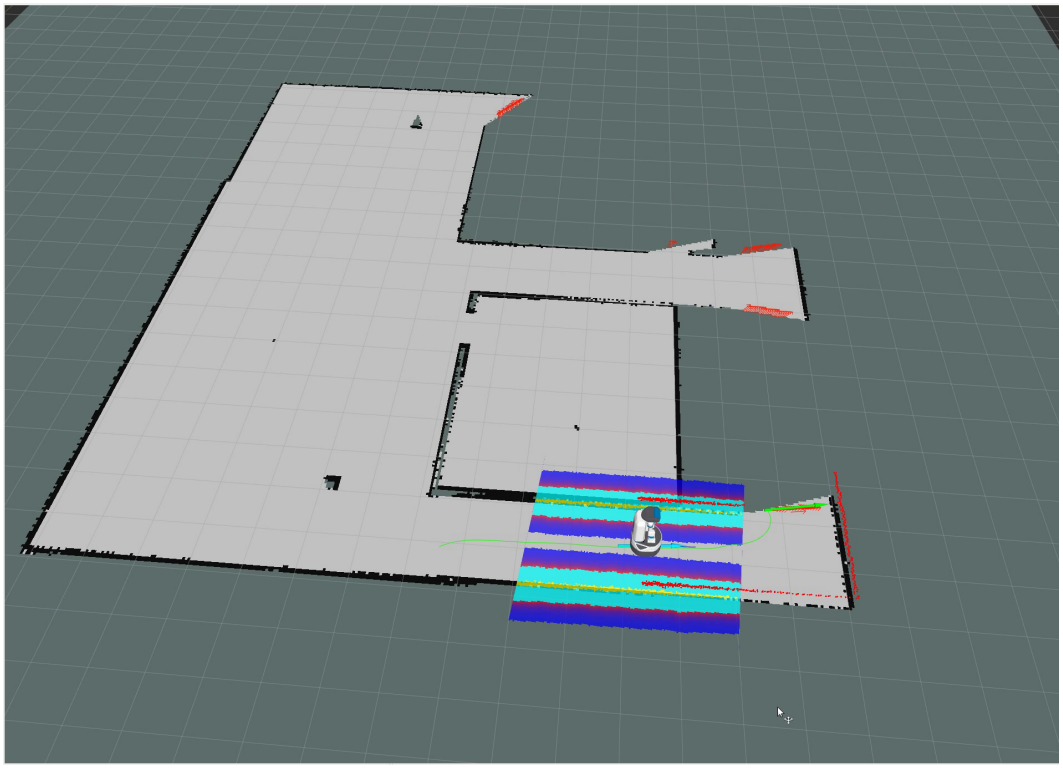


Figure 20. Robot continued its journey to uncover more corridor area

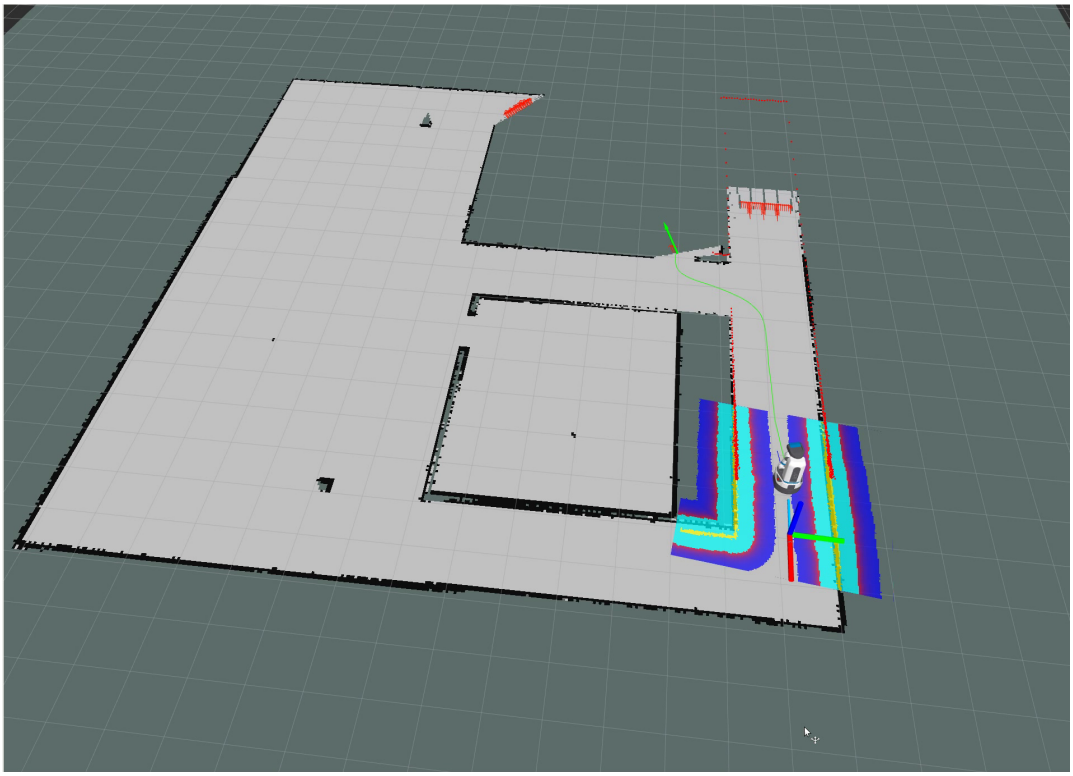


Figure 21. The robot planned to visit the second room of this simulation environment

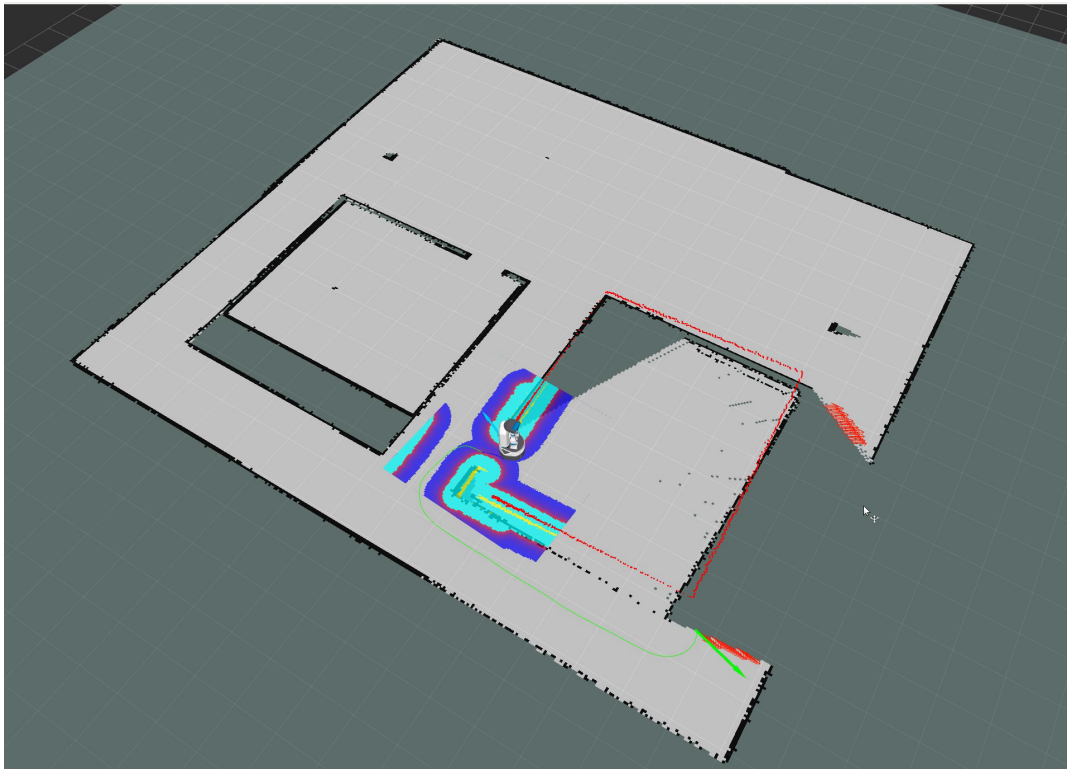


Figure 22. After the room has been visited, the robot was heading to the only unknown corridor

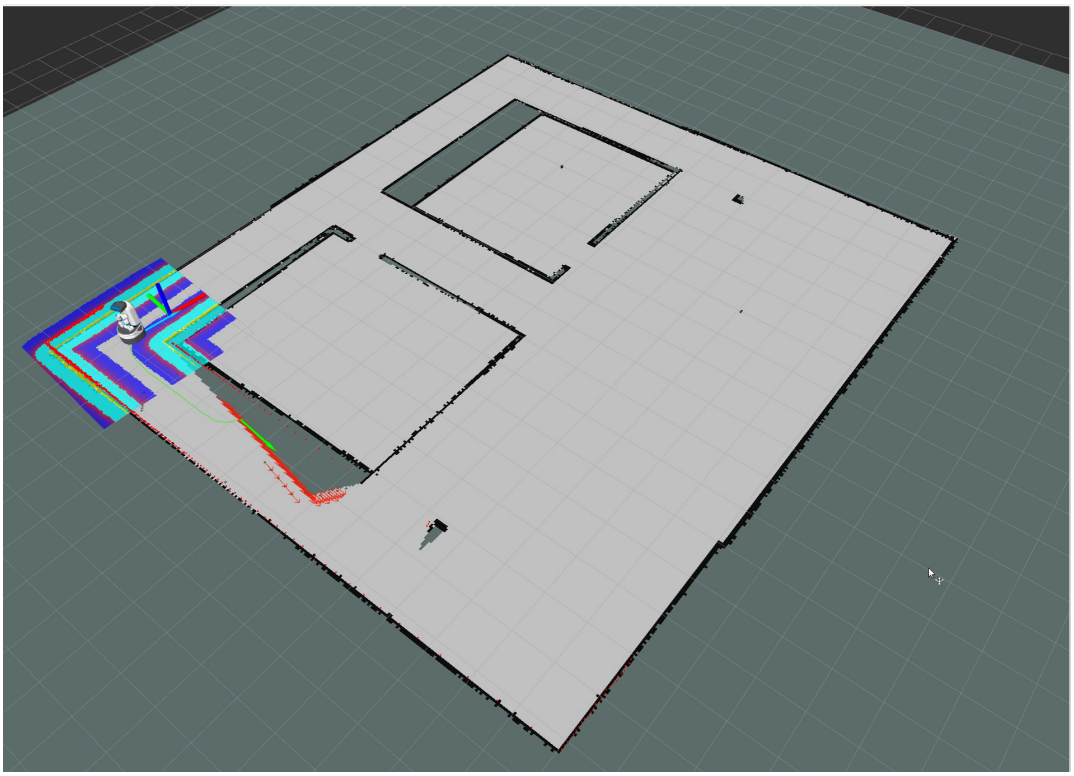


Figure 23. The last unknown part of this simulation environment will be visited by the robot

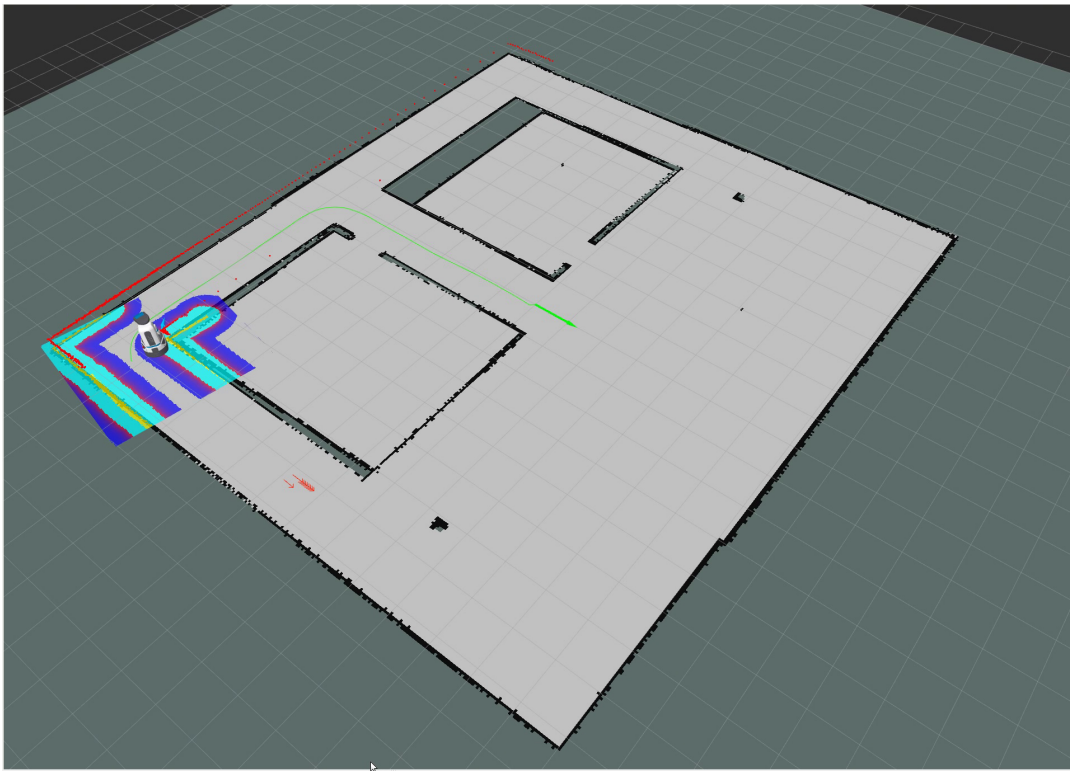


Figure 24. It was finished. Now, the robot is heading back to its starting point

After various successful simulation runs, this project has been approved by the project supervisors to test on real robot in actual environment populated with people.

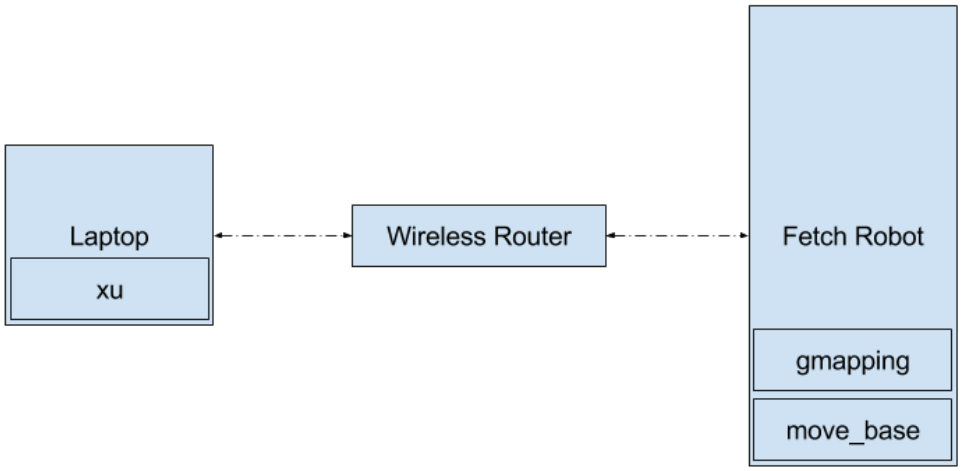


Figure 25. Software and network configuration for real environment test

The software and network settings diagram is shown in Figure 16 above. The exploration planner is named xu running on provided laptop connected to Fetch robot via a wireless router. The reset components SLAM and navigation nodes are operating directly in onboard computer inside the Fetch robot. Due to the publisher-subscriber architecture of ROS, all software components are not necessary running on the same machine. Therefore, the exploration planner is communicating with both SLAM and navigation module remotely.

Although the solution produced from this project runs quite well, there are still several inadequacies that needs to be mentioned below.

Figure 26. Fetch robot performing backward movement to unstuck itself from local costmap

Above Figure 26 is showing the Fetch robot performing unstack behavior due to the wall aside of it. After moving backward to try to get away from current stuck local costmap, the robot hit the wall. This concludes the existing move_base unstack behaviors still need more tuning based on experimental outcomes.

Figure 27. Fetch robot having collision with wall due to moving backward

Furthermore, due to the use of 2D laser as the only SLAM sensor input, gmapping mark the area behind the glass wall as accessible open ground. As a result, both exploration and navigation planner decide to move the robot outside of the closure environment.

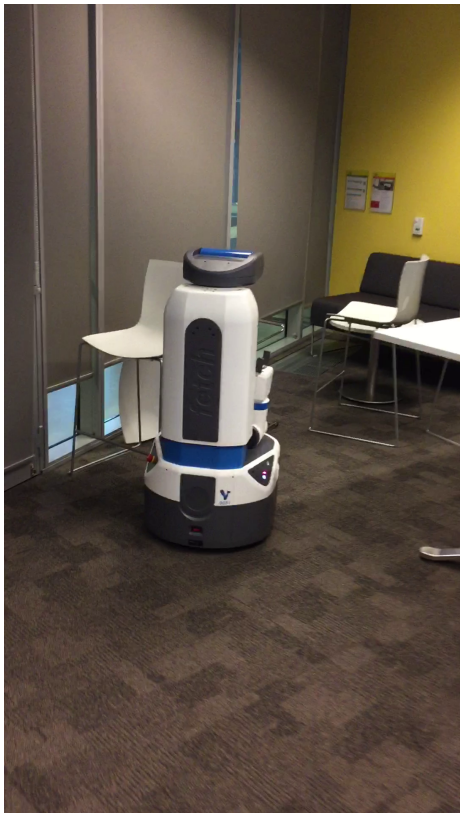


Figure 28. Fetch robot trying to move out of the room due to global path was planned out of the room caused by transparent glasses

Moreover, exploration planner occasionally crash during the second frontier planning process which was never happened in simulations. Fortunately, the exploration planning node is stateless therefore it didn't significantly affect the overall Active SLAM performance. The primary suspicion is the networking issue since it happens when the robot is far away from Wi-Fi router.

Foreseeable Enhancements

Through the observation during the real environment tests, it concludes following potential improvements:

- Use additional sensors for navigation and SLAM that will not affected by transparent objects such as ultrasonic sensor

- Tweaking move_base parameters to improve robot recovery behaviors from stuck state

- Adjust configurations in exploration planner to avoid oscillation between frontiers that far apart

- Profile the exploration planner on Fetch robot to debug the actual reasons that crash the node during the second frontier finding iteration

- Use machine learning in exploration strategy to find better goals accomplish different scenarios such as rescue mission after domestic nature disaster and interactive games like Peekaboo

Conclusion

In conclusion, using state-of-art SLAM and navigation package with homemade exploration planner in this project has achieved its specified goals in real environment tests. Although the solution has its flaws, overall it was well performed in both simulation and real world test run. Moreover, it is vital to enhance its current performance even after this project has been concluded.

Reference

Fetch Robotics Inc, 2015, 'Robot Hardware'
http://docs.fetchrobotics.com/robot_hardware.html

Fetch Robotics Inc, 2015, 'Tutorial: Navigation'
http://docs.fetchrobotics.com/navigation.html

Grisetti, G. Stachniss, C. & Burgard, W., 'GMapping', viewed 10 October 2016
http://wiki.ros.org/gmapping

Lu, D.V. Ferguson, M. & Marder-Eppstein, E., 'move_base' viewed 10 October 2016
http://wiki.ros.org/move_base

Fetch Robotics Inc, 2015, 'tilt_head.py' viewed 10 October 2016
https://github.com/fetchrobotics/fetch_ros/blob/indigo-devel/fetch_navigation/scripts/tilt_head.py

APPENDIX

APPENDIX – Launch file used for simulation

```
<launch>
  <arg name="use_sim_time" default="true" />
  <arg name="droid" default="fetch" />
  <arg name="output" default="screen" />
  <arg name="use_gmapping" default="true" />
  <arg name="use_rviz" default="true"/>
  <arg name="use_move_base" default="true" />
  <arg name="use_xu" default="true"/>

  <!-- Navigation parameter files -->
  <arg name="move_base_include" default="$(find
fetch_navigation)/launch/include/move_base.launch.xml" />

  <param name="use_sim_time" value="$(arg use_sim_time)" />

  <node if="$(arg use_gmapping)" pkg="gmapping" type="slam_gmapping"
name="slam_gmapping" output="$(arg output)">
    <remap from="scan" to="base_scan"/>
    <remap from="map" to="map"/>
    <param name="map_update_interval" value="1.0"/>

    <param name="maxUrange" value="10.0"/>
    <param name="maxRange" value="25.0"/>
    <param name="xmin" value="-5.0"/>
    <param name="xmax" value="5.0"/>
    <param name="ymin" value="-5.0"/>
    <param name="ymax" value="5.0"/>
  </node>

  <!-- move the robot -->
  <include if="$(arg use_move_base)" file="$(arg move_base_include)" >
    <arg name="name" value="$(arg droid)" />
  </include>

  <!-- tilt the head -->
  <node pkg="fetch_navigation" type="tilt_head.py" name="tilt_head_node" />

  <!-- RVIZ to view the visualization -->
        <node if="$(arg use_rviz)" name="RVIZ" pkg="rviz" type="rviz" args="-d $(find
xu)/config/xu.rviz" />
```

```
  <node if="$(arg use_xu)" name="xu" pkg="xu" type="xu" output="$(arg output)"
respawn="true">
    <param name="navigator" type="string" value="move_base"/>
    <param name="grid_resolution" type="double" value="1.0"/>
    <param name="origin_offset" type="double" value="0.8"/>
    <param name="exploration_strategy" type="string" value="nearest_first"/>
    <param name="midway_factor" type="double" value="1.0"/>
    <param name="send_goal" type="bool" value="true"/>
  </node>

</launch>
```

APPENDIX – Package definition file

```
<?xml version="1.0"?>
<package>
  <name>xu</name>
  <version>0.0.0</version>
  <description>The explorer unit package</description>

  <maintainer email="xw901103@gmail.com">Xu</maintainer>

  <!-- One license tag required, multiple allowed, one license per tag -->
  <!-- Commonly used license strings: -->
  <!--   BSD, MIT, Boost Software License, GPLv2, GPLv3, LGPLv2.1, LGPLv3 -->
  <license>TODO</license>

  <!-- Url tags are optional, but mutiple are allowed, one per tag -->
  <!-- Optional attribute type can be: website, bugtracker, or repository -->
  <!-- Example: -->
  <!-- <url type="website">http://wiki.ros.org/xu</url> -->

  <author email="xw901103@gmail.com">Xu</author>

  <buildtool_depend>catkin</buildtool_depend>
  <build_depend>boost</build_depend>
  <build_depend>geometry_msgs</build_depend>
  <build_depend>libopencv-dev</build_depend>
  <build_depend>message_filters</build_depend>
  <build_depend>nav_msgs</build_depend>
  <build_depend>roscpp</build_depend>
  <build_depend>roslib</build_depend>
  <build_depend>rospy</build_depend>
  <build_depend>sensor_msgs</build_depend>
  <build_depend>std_msgs</build_depend>
  <build_depend>tf</build_depend>
  <run_depend>boost</run_depend>
  <run_depend>geometry_msgs</run_depend>
  <run_depend>libopencv-dev</run_depend>
  <run_depend>message_filters</run_depend>
  <run_depend>nav_msgs</run_depend>
  <run_depend>roscpp</run_depend>
  <run_depend>roslib</run_depend>
  <run_depend>rospy</run_depend>
  <run_depend>sensor_msgs</run_depend>
  <run_depend>std_msgs</run_depend>
  <run_depend>tf</run_depend>
```

```
  <!-- The export tag contains other, unspecified, tags -->
  <export>
    <!-- Other tools can request additional information be placed here -->

  </export>
</package>
```

APPENDIX – CMakeLists of exploration planner package

```
cmake_minimum_required(VERSION 2.8.3)
project(xu)

## Find catkin macros and libraries
## if COMPONENTS list like find_package(catkin REQUIRED COMPONENTS xyz)
## is used, also find other catkin packages
find_package(catkin REQUIRED COMPONENTS
  geometry_msgs
  message_filters
  nav_msgs
  roscpp
  roslib
  rospy
  sensor_msgs
  std_msgs
  tf
)

## System dependencies are found with CMake's conventions
find_package(Boost REQUIRED COMPONENTS system)
find_package(OpenCV REQUIRED)

## Uncomment this if the package has a setup.py. This macro ensures
## modules and global scripts declared therein get installed
## See http://ros.org/doc/api/catkin/html/user_guide/setup_dot_py.html
# catkin_python_setup()

################################################
## Declare ROS messages, services and actions ##
################################################

## To declare and build messages, services or actions from within this
## package, follow these steps:
## * Let MSG_DEP_SET be the set of packages whose message types you use in
##   your messages/services/actions (e.g. std_msgs, actionlib_msgs, ...).
## * In the file package.xml:
##   * add a build_depend tag for "message_generation"
##   * add a build_depend and a run_depend tag for each package in MSG_DEP_SET
##   * If MSG_DEP_SET isn't empty the following dependency has been pulled in
##     but can be declared for certainty nonetheless:
##     * add a run_depend tag for "message_runtime"
## * In this file (CMakeLists.txt):
##   * add "message_generation" and every package in MSG_DEP_SET to
```

```
##    find_package(catkin REQUIRED COMPONENTS ...)
##   * add "message_runtime" and every package in MSG_DEP_SET to
##    catkin_package(CATKIN_DEPENDS ...)
##   * uncomment the add_*_files sections below as needed
##    and list every .msg/.srv/.action file to be processed
##   * uncomment the generate_messages entry below
##   * add every package in MSG_DEP_SET to generate_messages(DEPENDENCIES ...)

## Generate messages in the 'msg' folder
# add_message_files(
#   FILES
#   Message1.msg
#   Message2.msg
# )

## Generate services in the 'srv' folder
# add_service_files(
#   FILES
#   Service1.srv
#   Service2.srv
# )

## Generate actions in the 'action' folder
# add_action_files(
#   FILES
#   Action1.action
#   Action2.action
# )

## Generate added messages and services with any dependencies listed here
# generate_messages(
#   DEPENDENCIES
#   geometry_msgs#   nav_msgs#   sensor_msgs#   std_msgs
# )

################################################
## Declare ROS dynamic reconfigure parameters ##
################################################

## To declare and build dynamic reconfigure parameters within this
## package, follow these steps:
## * In the file package.xml:
##   * add a build_depend and a run_depend tag for "dynamic_reconfigure"
## * In this file (CMakeLists.txt):
```

```
##   * add "dynamic_reconfigure" to
##     find_package(catkin REQUIRED COMPONENTS ...)
##   * uncomment the "generate_dynamic_reconfigure_options" section below
##     and list every .cfg file to be processed

## Generate dynamic reconfigure parameters in the 'cfg' folder
# generate_dynamic_reconfigure_options(
#   cfg/DynReconf1.cfg
#   cfg/DynReconf2.cfg
# )

###################################
## catkin specific configuration ##
###################################
## The catkin_package macro generates cmake config files for your package
## Declare things to be passed to dependent projects
## INCLUDE_DIRS: uncomment this if you package contains header files
## LIBRARIES: libraries you create in this project that dependent projects also need
## CATKIN_DEPENDS: catkin_packages dependent projects also need
## DEPENDS: system dependencies of this project that dependent projects also need
catkin_package(
  INCLUDE_DIRS include
  LIBRARIES ${PROJECT_NAME}
  CATKIN_DEPENDS geometry_msgs message_filters nav_msgs roscpp roslib rospy
sensor_msgs std_msgs tf
  DEPENDS OpenCV
)

###########
## Build ##
###########

## Specify additional locations of header files
## Your package locations should be listed before other locations
# include_directories(include)
include_directories(
  include
  ${catkin_INCLUDE_DIRS}
  ${OpenCV_INCLUDE_DIRS}
)

## Declare a C++ library

## Add cmake target dependencies of the library
```

```
## as an example, code may need to be generated before libraries
## either from message generation or dynamic reconfigure
# add_dependencies(xu ${${PROJECT_NAME}_EXPORTED_TARGETS}
${catkin_EXPORTED_TARGETS})

## Declare a C++ executable
add_executable(${PROJECT_NAME} src/xu.cpp)

## Add cmake target dependencies of the executable
## same as for the library above
add_dependencies(${${PROJECT_NAME}_EXPORTED_TARGETS}
${catkin_EXPORTED_TARGETS})

## Specify libraries to link a library or executable target against
target_link_libraries(
  ${PROJECT_NAME}
  ${catkin_LIBRARIES}
  ${OpenCV_LIBRARIES}
)

#############
## Install ##
#############

# all install targets should use catkin DESTINATION variables
# See http://ros.org/doc/api/catkin/html/adv_user_guide/variables.html

## Mark executable scripts (Python etc.) for installation
## in contrast to setup.py, you can choose the destination
# install(PROGRAMS
#   scripts/my_python_script
#   DESTINATION ${CATKIN_PACKAGE_BIN_DESTINATION}
# )

## Mark executables and/or libraries for installation
# install(TARGETS xu xu_node
#   ARCHIVE DESTINATION ${CATKIN_PACKAGE_LIB_DESTINATION}
#   LIBRARY DESTINATION ${CATKIN_PACKAGE_LIB_DESTINATION}
#   RUNTIME DESTINATION ${CATKIN_PACKAGE_BIN_DESTINATION}
# )

## Mark cpp header files for installation
# install(DIRECTORY include/${PROJECT_NAME}/
#   DESTINATION ${CATKIN_PACKAGE_INCLUDE_DESTINATION}
```

```
#   FILES_MATCHING PATTERN "*.h"
#   PATTERN ".svn" EXCLUDE
# )

## Mark other files for installation (e.g. launch and bag files, etc.)
# install(FILES
#   # myfile1
#   # myfile2
#   DESTINATION ${CATKIN_PACKAGE_SHARE_DESTINATION}
# )

##############
## Testing ##
##############

## Add gtest based cpp test target and link libraries
# catkin_add_gtest(${PROJECT_NAME}-test test/test_xu.cpp)
# if(TARGET ${PROJECT_NAME}-test)
#   target_link_libraries(${PROJECT_NAME}-test ${PROJECT_NAME})
# endif()

## Add folders to be run by python nosetests
# catkin_add_nosetests(test)
```

APPENDIX – Exploration planner source code

```
#include <ros/ros.h>
#include <nav_msgs/GetMap.h>
#include <tf/transform_listener.h>
#include <actionlib/client/simple_action_client.h>
#include <move_base_msgs/MoveBaseAction.h>
#include <geometry_msgs/PoseStamped.h>
#include <geometry_msgs/PoseArray.h>
#include <boost/random.hpp>

#define PI 3.14159265

template <typename T>
class Vector2D {
public:
 inline Vector2D() {
 }
 inline explicit Vector2D(T _x, T _y, T _o): x(_x), y(_y), o(_o) {
 }
 inline Vector2D(const Vector2D<T>& ref): x(ref.x), y(ref.y), o(ref.o) { /* copy constructor */
 }

 inline Vector2D<T>& operator =(const Vector2D<T>& ref) { /* assign operator overload */
  this->x = ref.x;
  this->y = ref.y;
  this->o = ref.o;
  return *this;
 }

 inline bool operator ==(const Vector2D<T>& ref) const { /* equal operator overload */
  return this->x == ref.x && this->y == ref.y && this->o == ref.o;
 }
 inline bool operator !=(const Vector2D<T>& ref) const {
  return this->x != ref.x || this->y != ref.y || this->o != ref.o;
 }

 inline double calculateDistance(const Vector2D<T>& ref) const { /* calculate distance
between two points */
  return sqrt(pow(this->x - ref.x, 2) + pow(this->y - ref.y, 2));
 }

 T x;
 T y;
 T o;
```

```
};

typedef Vector2D<int> Grid;
typedef Vector2D<double> Frontier;

class XU {
  ros::NodeHandle nodeHandle;
  tf::TransformListener tfListener;
  actionlib::SimpleActionClient<move_base_msgs::MoveBaseAction> moveBaseActionClient;
  move_base_msgs::MoveBaseGoal moveBaseGoal;
  boost::thread processThread;
  ros::Rate rate;
  bool explorationFinished;
  bool moveBaseGoalSent;
  ros::ServiceClient getMapClient;
  nav_msgs::OccupancyGrid map;
  Vector2D<double> location;
  Vector2D<double> origin;
  std::vector<Grid> movements;
  std::vector<Frontier> frontiers;
  ros::Publisher locationPublisher;
  ros::Publisher originPublisher;
  ros::Publisher goalPublisher;
  ros::Publisher frontiersPublisher;
  ros::Publisher mapPublisher;
  std::string explorationStrategy;
  std::string navigator;
  bool sendGoal;
  double midwayFactor;
  double gridResolution;
  double originOffset;
public:
  XU(const ros::NodeHandle& nodeHandle): nodeHandle(nodeHandle),
tfListener(ros::Duration(1.0)), moveBaseActionClient("move_base", true), rate(1.0),
moveBaseGoalSent(false), gridResolution(1.0), originOffset(1.0) {
    this->nodeHandle.param<std::string>( "navigator", this->navigator, "move_base");
    this->nodeHandle.param<bool>( "send_goal", this->sendGoal, true);
    this->nodeHandle.param<std::string>( "exploration_strategy", this->explorationStrategy,
"nearest_first");
    this->nodeHandle.param<double>( "grid_resolution", this->gridResolution, 1.0);
    this->nodeHandle.param<double>( "origin_offset", this->originOffset, 1.0);
    this->nodeHandle.param<double>( "midway_factor", this->midwayFactor, 0.5);

    movements.push_back(Grid(-1 / this->gridResolution, -1 / this->gridResolution, 0));
```

```
    movements.push_back(Grid(-1 / this->gridResolution, 1 / this->gridResolution, 0));
    movements.push_back(Grid(1 / this->gridResolution, 1 / this->gridResolution, 0));
    movements.push_back(Grid(1 / this->gridResolution, -1 / this->gridResolution, 0));
    movements.push_back(Grid(-1 / this->gridResolution, 0, 0));
    movements.push_back(Grid(1 / this->gridResolution, 0, 0));
    movements.push_back(Grid(0, -1 / this->gridResolution, 0));
    movements.push_back(Grid(0, 1 / this->gridResolution, 0));

    this->locationPublisher =
this->nodeHandle.advertise<geometry_msgs::PoseStamped>("/xu/location", 1);
    this->originPublisher =
this->nodeHandle.advertise<geometry_msgs::PoseStamped>("/xu/origin", 1);
    this->goalPublisher =
this->nodeHandle.advertise<geometry_msgs::PoseStamped>("/xu/goal", 1);
    this->frontiersPublisher =
this->nodeHandle.advertise<geometry_msgs::PoseArray>("/xu/frontiers", 1);
    this->mapPublisher = this->nodeHandle.advertise<nav_msgs::OccupancyGrid>("/xu/map",
1, true);

    this->explorationFinished = false;

    this->getMapClient =
this->nodeHandle.serviceClient<nav_msgs::GetMap>("/dynamic_map");
    this->processThread = boost::thread(boost::bind(&XU::process, this));
  }

  void localize() {
    try {
      tf::StampedTransform transform;

      // transform the base_link tf back to map to obtain current estimated location of robot
      this->tfListener.lookupTransform("map", "base_link", ros::Time(0), transform);
      tf::Quaternion q = transform.getRotation();
      tf::Vector3 v = transform.getOrigin();
      this->location = Vector2D<double>(v.x(), v.y(), q.getAngle());

      geometry_msgs::PoseStamped locationMessage;
      locationMessage.header.stamp = ros::Time::now();
      locationMessage.header.frame_id = "map";
      tf::poseTFToMsg(
        tf::Pose(
          tf::createQuaternionFromYaw(this->location.o),
          tf::Vector3(this->location.x, this->location.y, 0.0)
        ),
```

```
      locationMessage.pose
    );
    this->locationPublisher.publish(locationMessage);

    ROS_INFO("[XU] slam odom x: %f y: %f orientation: %f", this->location.x, this->location.y,
this->location.o);
   } catch (tf::TransformException e) {
    ROS_ERROR("[XU] %s",e.what());
   }
  }

  void process() {
   if (this->navigator == "move_base") {
    if (!this->moveBaseActionClient.waitForServer()) {
     ROS_ERROR("[XU] move_base doesn't exist?");
     return;
    }
   }

   while(this->nodeHandle.ok()) {

    this->localize();

    if(!this->getMapClient.isValid()) {
                    ROS_ERROR("[XU] GetMap-Client is invalid!");
    } else {
     nav_msgs::GetMap getMap;
     if(!this->getMapClient.call(getMap)) {
      ROS_INFO("[XU] could not get a map.");
     } else {
      this->map = getMap.response.map;
      this->processMap();

      if (this->isFrontierInvalid()) {
       ROS_INFO("[XU] frontier will be canceled due to invalid");
       this->moveBaseGoalSent = false;
      } else if (this->isFrontierUnknow() == false) { /* frontier is known so it will start a new
one */
       this->moveBaseGoalSent = false;

      }

      if (this->navigator == "nav2d" && this->moveBaseGoalSent) {
```

```
          double distance = sqrt(pow(this->moveBaseGoal.target_pose.pose.position.x -
this->location.x, 2) + pow(this->moveBaseGoal.target_pose.pose.position.y - this->location.y,
2));
           this->moveBaseGoalSent = distance > 1.0;
         }

         if (this->moveBaseGoalSent == false) { /* start to find a new goal for the robot */
           ROS_INFO("[XU] prepare new goal");
           this->moveBaseGoal.target_pose = this->generateFrontierPose();

           if (this->sendGoal && this->frontiers.empty() == false) {
             if (this->navigator == "move_base") {
               this->moveBaseActionClient.cancelAllGoals();
               this->moveBaseActionClient.sendGoal(this->moveBaseGoal,
boost::bind(&XU::moveBaseDoneCallback, this, _1, _2),
boost::bind(&XU::moveBaseActiveCallback, this),
boost::bind(&XU::moveBaseFeedbackCallback, this, _1));
             }
           } else if (this->explorationFinished == false) { // move robot to 0, 0 on known map
             ROS_INFO("[XU] no more frontiers, go back 0, 0");
             if (this->navigator == "move_base") {
               this->moveBaseActionClient.cancelAllGoals();
               this->moveBaseActionClient.sendGoal(this->moveBaseGoal,
boost::bind(&XU::moveBaseDoneCallback, this, _1, _2),
boost::bind(&XU::moveBaseActiveCallback, this),
boost::bind(&XU::moveBaseFeedbackCallback, this, _1));
             }
             this->explorationFinished = true;
           }

           this->goalPublisher.publish(this->moveBaseGoal.target_pose);
           this->moveBaseGoalSent = true;
         }
       }
     }

     this->rate.sleep();
   }
   ROS_INFO("[XU] process dropped");
  }

  void moveBaseDoneCallback(const actionlib::SimpleClientGoalState& state, const
move_base_msgs::MoveBaseResultConstPtr& result) { /* goal is reached */
     this->moveBaseGoalSent = false;
```

```
  }

  void moveBaseActiveCallback() { /* goal is now in active */
    this->moveBaseGoalSent = true;
  }

  void moveBaseFeedbackCallback(const move_base_msgs::MoveBaseFeedbackConstPtr&
feedback) {

   actionlib::SimpleClientGoalState state = this->moveBaseActionClient.getState();

   if(state != actionlib::SimpleClientGoalState::ACTIVE && state !=
actionlib::SimpleClientGoalState::PENDING) {
     ROS_INFO("goal state is not ACTIVE or PENDING, current state %s",
this->moveBaseActionClient.getState().toString().c_str());
     //this->moveBaseActionClient.cancelAllGoals();
     this->moveBaseGoalSent = false;
     return;
   }

   if(this->map.info.width == 0 || this->map.info.height == 0) {
          ROS_INFO("[XU] Could not get a map.");
   } else {
     if(this->moveBaseGoalSent) {
     }
   }
}

  void processMap() {
   if(this->map.info.width == 0 || this->map.info.height == 0) {
          ROS_INFO("[XU] could not get a map.");
   } else {
     int width = this->map.info.width;
     int height = this->map.info.height;
     int inflation = static_cast<int>(1.0 / this->map.info.resolution);

     for (int y = 0; y < height; ++y) {
       for (int x = 0; x < width; ++x) {
         int index = y * width + x;
         if (this->map.data[index] == 100) { //inflate
           for (int o = 0; o < 360; ++o) { /* start inflate the cell 360 degree around it */
             double radian = o * PI / 180.0;
             for (int r = 1; r < 10; ++r) {
               int iy = static_cast<int>(y + r * sin(radian));
```

```
            int ix = static_cast<int>(x + r * cos(radian));
            //ROS_INFO("%d %d %d from %d %d", o, ix, iy, x, y);
            int ii = iy * width + ix;
            if (ii > 0 && ii < this->map.data.size() && this->map.data[ii] != 100) {
              this->map.data[ii] = 50;
            }
           }
          }
         }
        }
       }
    for (int y = 0; y < height; ++y) {
      for (int x = 0; x < width; ++x) {
        int index = y * width + x;
        if (this->map.data[index] == 50) { // dye color
          this->map.data[index] = 100;
        }
       }
    }

    this->mapPublisher.publish(this->map);
   }
 }

  void searchFrontiers() {
   if(this->map.info.width == 0 || this->map.info.height == 0) {
         ROS_INFO("[XU] could not get a map.");
   } else {
     double width = this->map.info.width;
     double height = this->map.info.height;
     double resolution = this->map.info.resolution;
     double originX = this->map.info.origin.position.x - (width * resolution + 2.0 *
this->map.info.origin.position.x); // idealy it should be zero for map origin offset
     double originY = this->map.info.origin.position.y - (height * resolution + 2.0 *
this->map.info.origin.position.y);
     ROS_INFO("[XU] map %ld width %f height %f resolution %f origin begin at %f %f",
this->map.data.size(), width, height, resolution, originX, originY);

     double offsetX = this->location.x + this->originOffset * cos(this->location.o);
     double offsetY = this->location.y + this->originOffset * sin(this->location.o);

     this->origin = Vector2D<double>(offsetX, offsetY, this->location.o);
     ROS_INFO("[XU] origin %f %f %f", this->origin.x, this->origin.y, this->origin.o);
```

```
    int xIndex = static_cast<int>((originX + offsetX) / resolution + width);
    int yIndex = static_cast<int>((originY + offsetY) / resolution + height);

    if (xIndex < 0 || yIndex < 0 || this->map.data[yIndex * width + xIndex] != 0) {
      // offset doesn't seems to be a valid or open cell, do the hard way to resolve frontiers
      for (int y = 0; y < height; ++y) {
        for (int x = 0; x < width; ++x) {
          if (this->map.data[y * width + x] == 0) {
            xIndex = x;
            yIndex = y;
            x = width;
            y = height;
          }
        }
      }
      this->origin.x = xIndex * resolution - originX + resolution / 2.0;
      this->origin.y = yIndex * resolution - originY + resolution / 2.0;
    }

    this->frontiers.clear();
    std::vector<Grid> openList;
    openList.push_back(Grid(xIndex, yIndex, 0));

    while(!openList.empty()) {
      Grid grid(openList.front());
      openList.erase(openList.begin());
      for(std::vector<Grid>::iterator iter = this->movements.begin(); iter !=
this->movements.end(); ++iter) {
        int index = (grid.y + iter->y) * width + (grid.x + iter->x);
        if (this->map.data[index] == -1) {
          this->frontiers.push_back(
            Vector2D<double>(
              (grid.x + iter->x - width) * resolution - originX + resolution / 2.0,
              (grid.y + iter->y - height) * resolution - originY + resolution / 2.0,
              atan2(
                //(grid.y + iter->y - height) * resolution - originY - this->origin.y,
                //(grid.x + iter->x - width) * resolution - originX - this->origin.x
                (grid.y + iter->y - height) * resolution - originY - this->location.y,
                (grid.x + iter->x - width) * resolution - originX - this->location.x
              )
            )
          );
          this->map.data[index] = -100; // close the grid
        } else if (this->map.data[index] == 0) { // open space
```

```
          openList.push_back(Grid(grid.x + iter->x, grid.y + iter->y, 0));
          this->map.data[index] = -100; // close the grid
        } else {
        }
      }
    }
    ROS_INFO("[XU] total frontiers %ld", this->frontiers.size());
    if (!this->frontiers.empty()) {
     this->publishFrontiers();
    }
   }
  }

  geometry_msgs::PoseStamped generateFrontierPose() {
   geometry_msgs::PoseStamped goalPose;
   double x = 0.0;
   double y = 0.0;
   double angle = 0.0;

   this->searchFrontiers();

   if (!this->frontiers.empty()) {
    if (this->explorationStrategy == "nearest_first") {
     double nearestDistance = std::numeric_limits<double>::max();
     double minimumDistance = 4.0;
     for (std::vector<Frontier>::iterator iter = this->frontiers.begin(); iter !=
this->frontiers.end(); ++iter) {
      double distance = iter->calculateDistance(this->location);
      if (distance > minimumDistance) {
       if (distance < nearestDistance) {
        x = iter->x;
        y = iter->y;
        angle = iter->o;
        nearestDistance = distance;
       }
      }
     }

     double midwayDistance = nearestDistance * this->midwayFactor;

     ROS_INFO("[XU] nearest_first distance %f", nearestDistance);
    } else if (this->explorationStrategy == "farthest_first") {
     double farthestDistance = 0.0;
```

```
      for (std::vector<Frontier>::iterator iter = this->frontiers.begin(); iter !=
this->frontiers.end(); ++iter) {
        double distance = iter->calculateDistance(this->origin);
        if (distance > farthestDistance) {
          x = iter->x;
          y = iter->y;
          angle = iter->o;
          farthestDistance = distance;
        }
      }

      double midwayDistance = farthestDistance * this->midwayFactor;
      x = this->origin.x + midwayDistance * cos(angle);
      y = this->origin.y + midwayDistance * sin(angle);

      ROS_INFO("[XU] farthest_first distance %f", farthestDistance);
    } else if (this->explorationStrategy == "midway") {
  } else {
      ROS_ERROR("[XU] unknow exploration strategy %s", this->explorationStrategy.c_str());
    }
  }

  goalPose.header.stamp = ros::Time::now();
  goalPose.header.frame_id = "/map";
  tf::poseTFToMsg(
    tf::Pose(tf::createQuaternionFromYaw(angle),
    tf::Vector3(x, y, 0.0)),
    goalPose.pose
  );
  ROS_INFO("[XU] goal pose x: %f y: %f o: %f", x, y, angle);

  return goalPose;
 }

 void publishFrontiers() {
  double width = this->map.info.width;
  double height = this->map.info.height;
  double resolution = this->map.info.resolution;
  double orginX = this->map.info.origin.position.x;
  double orginY = this->map.info.origin.position.y;

  geometry_msgs::PoseArray frontiersMessage;
  frontiersMessage.header.stamp = ros::Time::now();
  frontiersMessage.header.frame_id = "/map";
```

```
  frontiersMessage.poses.resize(this->frontiers.size());
  for (int i = 0; i < this->frontiers.size(); ++i) {

  tf::poseTFToMsg(
     tf::Pose(
       tf::createQuaternionFromYaw(this->frontiers[i].o),
       tf::Vector3(this->frontiers[i].x, this->frontiers[i].y, 0.0)
     ),
     frontiersMessage.poses[i]
    );
  }

  geometry_msgs::PoseStamped originMessage;
  originMessage.header.stamp = ros::Time::now();
  originMessage.header.frame_id = "map";
  tf::poseTFToMsg(
    tf::Pose(
      tf::createQuaternionFromYaw(this->origin.o),
      tf::Vector3(this->origin.x, this->origin.y, 0.0)
    ),
    originMessage.pose
  );
  this->originPublisher.publish(originMessage);
  this->frontiersPublisher.publish(frontiersMessage);
 }

 bool isFrontierUnknow() const {
  bool result = true;
  int xIndex = static_cast<int>((this->map.info.origin.position.x +
this->moveBaseGoal.target_pose.pose.position.x) / this->map.info.resolution) +
this->map.info.width;
  int yIndex = static_cast<int>((this->map.info.origin.position.y +
this->moveBaseGoal.target_pose.pose.position.y) / this->map.info.resolution) +
this->map.info.height;

  if (this->map.data[yIndex * this->map.info.width + xIndex] != -1) {
    ROS_INFO("[XU] frontier uncovered");
    result = false;
  }
  return result;
 }

 bool isFrontierInvalid() const {
  bool result = false;
```

```
    int xIndex = static_cast<int>((this->map.info.origin.position.x +
this->moveBaseGoal.target_pose.pose.position.x) / this->map.info.resolution) +
this->map.info.width;
    int yIndex = static_cast<int>((this->map.info.origin.position.y +
this->moveBaseGoal.target_pose.pose.position.y) / this->map.info.resolution) +
this->map.info.height;

    if (this->map.data[yIndex * this->map.info.width + xIndex] == 100) {
      ROS_INFO("[XU] frontier invalid");
      result = true;
    }
    return result;
  }

};

int main(int argc, char** argv) {
  ros::init( argc, argv, "xu" );
  ros::NodeHandle nodeHandle("~");
  XU xu(nodeHandle);
  ros::spin();
  return 0;
}
```