NAVIGATION PROJECT

Software architecture for robotics

Nav-04

Professors: Fulvio Mastrogiovanni, Mohammad Alameh, Alessandro Carfi, Simone Maccio

Students: Julio Sebastian Sullca Trillo, Alluri Sai Krishna Rama Chander Raju, Nagalakunta Sumanth, Hanish MuthuRathinam Paraman

Content

1 Introduction

The aim of the project is to lead a robot from an origin point to an endpoint, avoiding all the obstacles in the way and trying to use the shortest path. To do that we use the information from lidar sensors to compute the odometry of the robot. The steps for the navigation are:

- Mapping: Moving through the environment robot identifies the place which is surrounding him. Mapping makes robot identify the environment.
- Localization: The robot computes his position; this means where it is and the orientation that it has. Localization answers the question “Where I am?”
- Path planning: Compute the shortest path to reach the endpoint. With path planning we answer the question “How I get the endpoint?”
- Obstacle avoidance: Using the local map the robot detects the obstacles that are close him to avoid them.

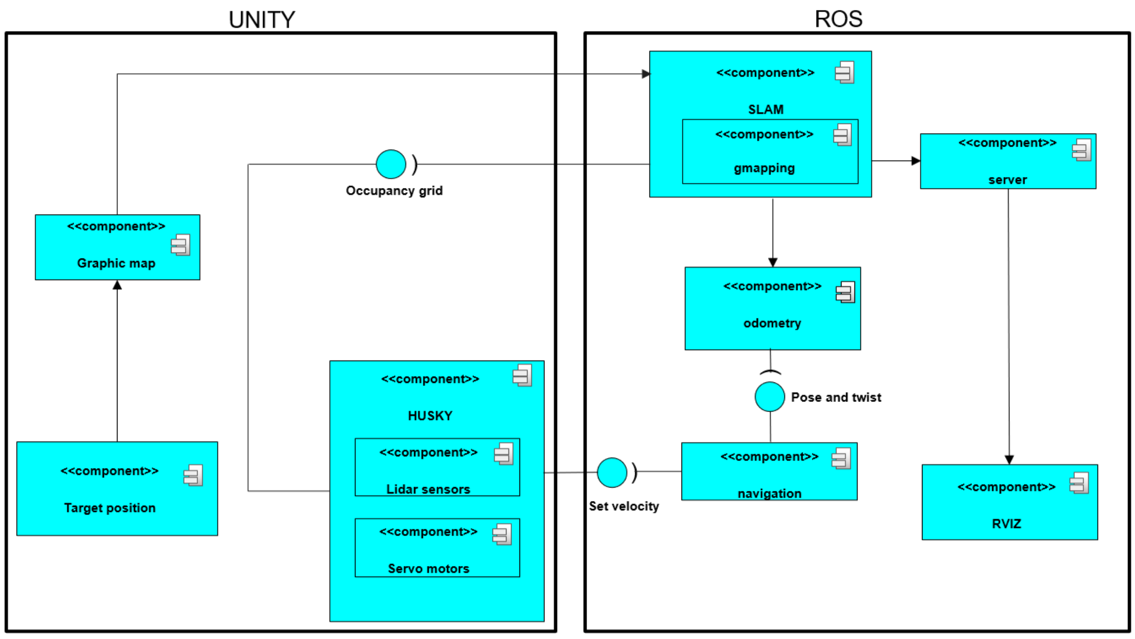
We create maps of environments, localize the robot in the environment, make the robots perform path planning, using SLAM (simultaneous localization and mapping). Regarding obstacle avoidance we can say that robot transform the whole map in discrete elements. This means that the maps are formed by many pieces that the robot can interpretate as empty or solid places.

These processes allow us to do the navigation just for a point, but we are trying to make the navigation for a robot, so we need to give information about our it to the system. These data are called Robot Configuration. For instance, in the Mapping system we will tell the system where the robot has the laser mounted on or his orientation this way, we will create an accurate map. Robot configuration and definition is done in the URDF files of the robot. Which is an XML format that describes a robot model. It defines its different parts, dimensions, kinematics, dynamics, sensors, etc...

The project is available in the following GitHub repository
https://github.com/sebastian97st/SoFar

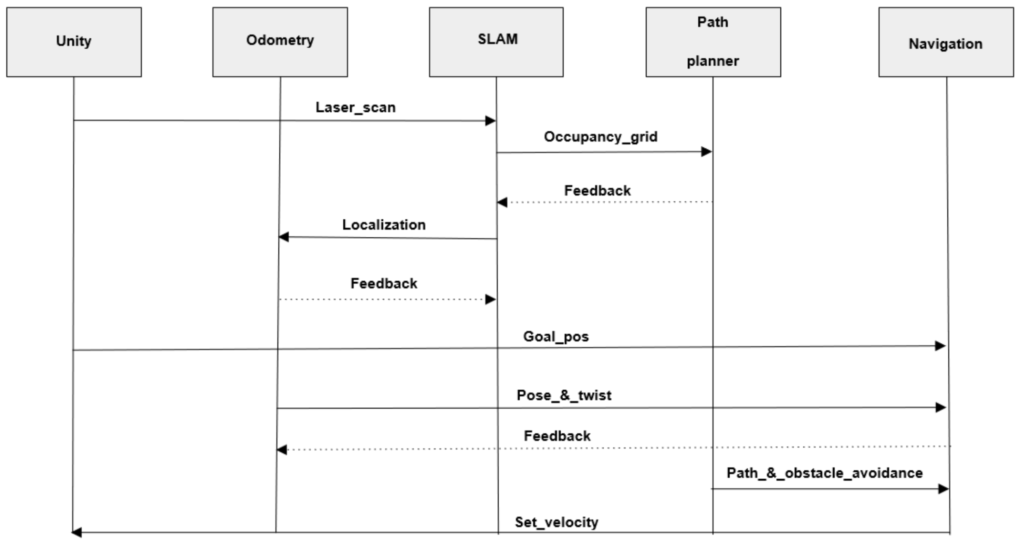
2 UML and time sequence diagram

UML



The graphic map and the occupancy grid (captured by lidar sensors) are sent from Unity to Ros, in order to SLAM software start working. Slam process the information and send it to the RVIZ software through a server. RVIZ will allow us seeing the map and path in the screen. Slam software is mapping the environment and computing the localization at the same time using the odometry part. Once we know the environment, the path, and our position we can compute the navigation part who is charged of sending the correspondent velocities to the servo motors simulation in Unity.

Time sequence diagram



3 Packages used in software

3.1 Gmapping

This package contains a ROS wrapper for OpenSlam's Gmapping. The Gmapping package provides laser-based SLAM (Simultaneous Localization and Mapping), as a ROS node called slam_gmapping. By using slam_gmapping, you can create a 2-Dimensional occupancy grid map (like a building floorplan) from laser and pose data collected by the sensor from a mobile robot.

3.2 Rviz

Rviz is a 3d representation apparatus for ROS applications. It gives a perspective on your robot model, catches sensor data from robot sensors, and replay caught information. It can show information from cameras, lasers, from 3D and 2D gadgets including pictures and point clouds. The robot state distributer assists you with broadcasting the state of your robot to the change library. The robot state distributer inside has a kinematic model of the robot; so given the joint places of the robot, the robot state distributer can register and communicate the 3D posture of each connection in the robot.

3.3 Move base

The move_base hub gives a ROS interface to arranging, running, and communicating with the route stack on a robot. Running the move_base hub on a robot that is appropriately arranged brings about a robot that will endeavour to accomplish an objective posture with its base to inside a client determined resistance. Without dynamic impediments, the move_base hub will ultimately get inside this resistance of its objective or sign inability to the client. The move_base hub may alternatively perform recuperation practices when the robot sees itself as stuck.

3.4 Planner

This shows the nearby cost map that the route stack utilizes for route. The yellow line is the distinguished impediment. For the robot to keep away from crash, the robot's impression ought to never cross with a cell that contains an impediment.

Cost map is everything the robot knows from past visits and put away information for example the guide. In the nearby cost map is all that can be known from the current situation with the sensors at this moment.

3.5 Navigation

The Navigation Stack is genuinely straight forward on an applied level. It learns from odometry, and sensor streams and yields speed orders to ship off a portable base. As essential for route stack the robot ought to have a transform function change tree set up and distribute sensor information utilizing the right ROS Message types.

4 Src contents

To make the robot move in a certain environment we have created the SoFar main package. Which contains 3 folders:

mobile_robot_navigation_project, husky and ros_tcp_endpoint

4.1 Mobile_robot_navigation_project: config. File

Contains all robot configuration and definition files.

4.1.1 model.rviz:

A pre-configured Rviz file.

4.1.2 Params.yaml:

Including Ros-Ip and Unity-Ip for the communication purposes

4.2 Mobile_robot_navigation_project: launch file

The dispatch bundle contains the roslaunch apparatuses, which peruses the roslaunch dispatch/XML design. It likewise contains an assortment of other help instruments to assist you with utilizing these documents. Numerous ROS bundles accompany "dispatch records", which you can run with:

We start the Ros simulation with the following file which allow us to localize the robot in the environment:

● roslaunch mobile_robot_navigation_project slamgmapping.launch

For launching the simulation of husky, simulation of the occupancy grid and path

● roslaunch mobile_robot_navigation_project rvizcontroller.launch

Then we start the communication between Ros and Unity:

● roslaunch mobile_robot_navigation_project navigation.launch

To finish we launch the following file for navigation, moving and path planning

● roslaunch mobile_robot_navigation_project movebase.launch

4.3 Mobile_robot_navigation_project: source file

This file contains the parameters which define the software performance.

4.3.1 Base_local_planner_src_param.yaml

The base_local_planner bundle gives a regulator that drives a versatile base in the plane. This regulator serves to associate the way organizer to the robot. Utilizing a guide, the organizer makes a kinematic direction for the robot to get from a beginning to an objective area. In route, the organizer makes, locally around the robot, a worth capacity, addressed as a matrix map. This worth capacity encodes the expenses of crossing through the lattice cells. The regulator's work is to utilize this worth capacity to decide dx, dy, dtheta speeds to ship off the robot.

The basic idea of both the Trajectory Rollout and Dynamic Window Approach (DWA) algorithms is as follows:

1. Discretely sample in the robot's control space (dx, dy, dtheta)

2. For each tested speed, perform forward reproduction from the robot's present status to foresee what might occur if the examined speed were applied for a few (brief) time frame.

3. Assess (score) every direction coming about because of the forward re-enactment, utilizing a metric that consolidates qualities, for example, nearness to obstructions, vicinity to the objective, vicinity to the worldwide way, and speed. Dispose of unlawful directions (those that impact with deterrents).

4. Pick the most noteworthy scoring direction and send the related speed to the versatile base.

5. Do this process again.

DWA varies from Trajectory Rollout in how the robot's control space is inspected. Direction Rollout tests from the arrangement of attainable speeds over the whole forward reproduction time frame given the speed increase cut-off points of the robot, while DWA tests from the arrangement of reachable speeds for only one re-enactment step given the speed increase cut-off points of the robot. This implies that DWA is a more effective calculation since it tests a more modest space, however, might be beaten by Trajectory Rollout for robots with low-speed increase limits in light of the fact that DWA doesn't advance mimic consistent speed increases. Practically speaking in any case, we discover DWA and Trajectory Rollout to perform equivalently in the entirety of our tests and suggest utilization of DWA for its effectiveness gains.

4.3.2 Costmap_common_src_param.yaml

The costmap_2D bundle gives a configurable design that keeps up with data about where the robot ought to explore as an inhabitancy matrix. The costmap utilizes sensor information also, data from the static guide to store and refresh data about obstructions on the planet through the costmap_2d::Costmap2DROS object.

4.3.3 Global_costmap_src_param.yaml

The "global_frame" boundary characterizes what organize outline the costmap should run in, in this case, we'll pick the/map outline. The "robot_base_frame" boundary characterizes the arrange outline the costmap should reference for the foundation of the robot.

4.3.4 Local_costmap_src_param.yaml

The "global_frame", "robot_base_frame", "update_frequency", and "static_map" boundaries are equivalent to depicted in the Global Configuration area above. The "publish_frequency" boundary decides the rate, in Hz, at which the costmap will distribute perception data. Setting the "rolling_window" boundary to genuine implies that the costmap will remain revolved around the robot as the robot travels through the world. The "width," "tallness," and "goal" boundaries set the width (meters), tallness (meters), and goal (meters/cell) of the costmap. Note that it's fine for the goal of this network to be not the same as the goal of your static guide, yet more often than not we will in general set them equivalent.

To score directions effectively, the Map Grid is utilized. For each control cycle, a matrix is made around the robot (the size of the nearby costmap), and the worldwide way is planned onto this region. This implies sure of the matrix cells will be set apart with distance 0 to a way point, and distance 0 to the objective. A spread calculation then, at that point effectively denotes any remaining cells with their Manhattan distance to the nearest of the focuses set apart with zero. This map matrix is then utilized in the scoring of directions

4.3.5 Move_base_src_param.yaml

The objective of the worldwide way may frequently lie outside the little region covered by the map_grid, when scoring directions for vicinity to the objective, what is considered is the "neighbourhood objective", which means the first way point which is inside the space having a sequential point outside the space. The size of the region is dictated by move_base.

4.4 Mobile_robot_navigation_project: scripts file

Contains two documents provided by professor.

4.4.1 Odometry_publisher.py

The route stack utilizes tf to decide the robot's area on the planet and relate sensor information to a static guide. Nonetheless, tf doesn't give any data about the speed of the robot. Along these lines, the route stack necessitates that any odometry source distribute both a change what's more, a nav_msgs/Odometry message over ROS that contains speed data. This instructional exercise clarifies the nav_msgs/Odometry message and gives model code to distributing both the message and change over ROS and tf individually.

4.4.2 Server_endpoint.py

This module really incorporates Establishing an association among ROS and solidarity utilizing change control convention .it Reads int32 from attachment association with decide the number of bytes to peruse to get the string that follows. Peruse that number of bytes and unravel to utf-8 string.it makes a Base class for ROS correspondence where information is shipped off the ROS network both inside and remotely. It additionally creates and ties attachments utilizing TCP factors then, at that point tunes in for approaching associations. For each new association a customer string will be made to deal with correspondence.

4.5 Mobile_robot_navigation_project: Husky file

Contains packages about Husky robot configuration and LMS1XX software. The packages can be downloaded from the followings GitHub link:

https://github.com/clearpathrobotics/lms1xx

https://github.com/husky/husky

5 Installation

To execute the simulation, we need two machines (or two virtual machines), one with a Windows OS and the other with a Linux OS. On Windows side should be acquainted Unity and relative frameworks with show the re-enactment, on Linux side should be acquainted ROS Noetic with run the modules.

5.1 UNITY INSTALLATION (ON WINDOWS)

1. Visit https://unity3d.com/get-unity/download and download Unity Hub
2. Through the Hub, install Unity 2020.2.2 (the version for which the projects have been developed) or later. Beware that later versions (e.g. Unity 2021.1.x) may be incompatible with the projects.
3. In order to download the pre required packages we have to go through this path https://github.com/TheEngineRoom-UniGe/SofAR-Mobile-Robot-Navigation then download the zip folder and extract it to the unity.

4. Visit to download the Unity project folder, extract it, then open Unity Hub and ADD the project to your projects list using the associated button.
5. Open the project.
6. In the bar on top of the screen, open the Robotics/ROS Settings tab and replace the ROS_IP with the IP of the machine running ROS.
7. If the previous steps have been successful, you should be able to enter Editor mode (via the Play button) and play the simulation.
8. If the communication is running correctly, on UBUNTU you can echo the topics that are being exchanged between ROS and Unity.

5.2 ROS NOETIC INSTALLATION (ON LINUX UBUNTU 20.04)

1. Install ROS Noetic following the tutorial on http://wiki.ros.org/noetic/Installation/Ubuntu.
2. Install the slam and navigation framework, visit http://wiki.ros.org/gmapping , https://wiki.ros.org/navigation.

   Install packages running the following commands in the Linux console:

   Cd/opt/ros/noetic/lib

   Sudo apt-get install ros-noetic-move-base-msgs

   Sudo apt-get install ros-noetic-move-base

   Sudo apt-get install ros-noetic-gmapping
3. Once all dependencies are resolved, do:
4. Create your ROS workspace.
5. Visit https://github.com/sebastian97st/SoFar and download the ROS packages needed for the project.
6. Extract the packages to your workspace.
7. Open the 'mobile_robot_navigation' package, then navigate to the config folder and open the params.yaml file. After ROS_IP, insert the IP address of your Linux machine and your Windows IP machine after UNITY_IP
8. Compile packages with catkin_make.
9. Once all dependencies are resolved, do:

● 'roslaunch mobile_robot_navigation_project slamgmapping.launch', to run localization package

● 'roslaunch mobile_robot_navigation_project rvizcontroller.launch', to launch RVIZ software

● 'roslaunch mobile_robot_navigation_project navigation.launch', o start communication with Unity (ensure the server communication is up and running, you should see the following line in the terminal ' Starting server on YOUR IP:10000'

● 'roslaunch mobile_robot_navigation_project movebase.launch', to navigate

TIPS:

1. If you experience a strange window when opening the Unity Project for the first time, simply do Quit, then reopen the project and it should not bother you further.
2. Should you experience issues in the communication between ROS and Unity, especially when publishing FROM ROS TO Unity, remember to disable the firewall of your PC, as it could interfere with the sending of messages over TCP.
3. The connection on the machines must be private and the type has to be BRIDGE.
4. When running your project, be sure to launch the ROS files BEFORE playing the Unity simulation, in order to have the server endpoint node up and running before initializing communication.

5.3 Code execution

Write in each console the follow line before launching source

devel/setup.bash

To launch localization

roslaunch mobile_robot_navigation_project slamgmapping.launch

To launch urdf of husky etc in rviz

roslaunch mobile_robot_navigation_project rvizcontroller.launch

To launch the handshake between unity and ros
roslaunch mobile_robot_navigation_project navigation.launch

To navigate
roslaunch mobile_robot_navigation_project movebase.launch

Then press play bottom in Unity to launch simulation, it’s recommended that RVIZ software be ready before the following step, pressing move button in Unity.

6 Test and Results

The parameter configuration of the robot can finish the path without collisions albeit now and again when the robot is drawing nearer to the end point, we see that the robot going through the end point or making circles around it and following couple of moments robot spans to the end point.

Regarding path planning the robot choose the shorter one, moreover we can see in the RVIZ software that the path and mapping is correct. Summing up the robot navigation procedure is:

First maps its surrounded environment and added to the global map. Then decide a correct path avoiding the obstacles, always looking for the shorter path to the endpoint. The mapping and pathing are run at the same time, because of the SLAM software.