UNIVERSITÀ
DEGLI STUDI
DI GENOVA

Software Architecture Project

Group ID: Nav-06

Project Members

Name	Student ID	Email Address
Omotoye Shamsudeen Adekoya	5066348	adekoyaomotoye@gmail.com
Farzin Sarvari	5057724	Sarvarifarzin94@gmail.com
Amanzhol Raisov	4889656	cornytravel@gmail.com
Taha Hussain Raja	5046306	rajagenoa@gmail.com

Mobile Robot Navigation and Mapping

Project Objectives

The aim of this project is to design and implement a ***software architecture*** for the control of a mobile robot *(Husky Robot)* to Navigate and Map an environment *(simulated or real-life)*. The Mobile Robot in question is a robot called **Husky Robot**, the robot is a *non-holonomic robot* equipped with a LaserScan sensor for detecting obstacle. A simulation environment is provided as a **Unity Scene** which is contained by the *husky robot* and an environment with obstacles to navigate through, there is also a goal point which the robot is intended to navigate to. The control of the mobile robot to perform the aforementioned task is going to be done through ROS packages, therefore the ROS packages and how they interact through interfaces to perform the goal is what this project is going to focus on (basically investing the software architecture design for the ROS packages).

Enumerated goals the software architecture is intended to achieve with the robot (simulated and real-life)

- Create a 3D Map (SLAM)
- Implement an Exploration Logic to reach a target in the environment.

For more information about the project requirements, Click Here.

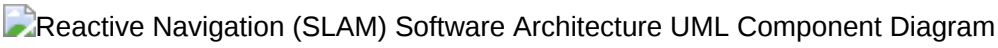
NOTE: ***some objectives has been changed because of simulation limitations***

System Architectures

Three different system architecture was implemented for this project each with their different pros and cons.

1. Reactive Navigation (SLAM), *Dijkstra's algorithm for Navigation*
2. MapBased Navigation, *Dijkstra's algorithm for Navigation*

Reactive Navigation (SLAM)


Reactive Navigation (SLAM) Software Architecture UML Component Diagram

The architecture implements SLAM (Simultaneous Localization and Mapping) to navigate the robot from its initial pose to the target location. The SLAM package creates a map while moving through the unknown environment and then from the map created by the SLAM package, the robot can localize itself and the use the Dijkstra's algorithm to figure out the best path to take to the goal point base on the knowledge gather from the map creation. The architecture is split into two main components; the **Unity Scene** component, where the robot that is being controlled is simulated and the **ROS** component where the control of the simulated robot is taking place.

Unity Scene Component

The unity scene component is a simulation environment that is very similar to *Gazebo* only that this Simulator is more general than Gazebo. Asides from simulating an environment for ROS development (which only just got supported recently), it is used for thing like multiplatform game development, VR and AR application development etc. In this component there are two subcomponents, the *mobile robot and the goal planner*. The unity scene pulishes the **Robot Pose** (`/odometry_frame`) and **Laser Sensor Reading** (`/laser_scan`) from the Robot Model in the scene. The component implement a Publisher interface to send a ROS message **LaserScan** through the `/laser_scan` topic and a **Pose** message through the `/odometry_frame` topic. The Goal planner of the unity scene uses a slightly different interface; it uses an ***actionlib client***. The two Interfaces only differ in concept because it also publishes the message just like a publisher, only that for the actionlib client, it follow the client server concept, in which the client sends a request to a server and expects a response from the server. The goal planner sends a **geometry_msgs/PoseStamped** through the `move_base_simple/goal` topic. The only input message to the Unity Scene is a **geometry_msgs/Twist** message through the `/cmd_vel` topic. This message is sent from the ROS component to control the velocity of the Mobile Robot for the navigation of the robot. The Unity Scene Simulation environment was developed by TheEngineRoom-Unige Lab for the purpose of this project, the task for us is to create and ROS architecture to interact with the simulation environment to perform specific task. So basically there's nothing to change from the Simulation other than interacting with it.

The ROS Component (Robot Operating System)

The ROS component consist of the seven sub-component as follow

- Odometry Publisher
- Husky Robot Description
- Joint State Publisher
- Robot State Publisher
- Slam-toolbox (SLAM)
- RViz (Data Visualization)
- MoveBase (Dijkstra's Navigation)

Odometry Publisher

This ROS node takes the *Robot Pose* message coming from `/odometry_frame` topic and use it to create a transformation between `/odom` and `/base_link`, it then takes the transformation message and sends it to the `/tf` topic.

Husky Robot Description

The Husky Robot Description is a **URDF (Universal Robot Description Format)** model which is a collection of files that describe a robot's physical description (*robot frame transformations, the physical appearance of the robot etc...*) to ROS. The component publishes this information into a ros parameter server for other node to read from when robot transformations and appearance is required. The component was gotten from a ros package from Husky Robot official Github Page that is added as a submodule to this github.

Joint State Publisher

This is a ROS inbuilt package that reads the robot_description parameter from the parameter server, finds all of the non-fixed joints and publishes a JointState message with all those joints defined. ROS Wiki. The JointState message is used by the Robot state publisher to determine the transformation between the robot joints.

Robot State Publisher

The is another ROS inbuild package that helps to publish the state of the robot to `/tf`. This package subscribes to joint states of the robot from the joint state publisher and publishes the 3D pose of each link using the kinematic representation from the URDF model. It publishes the state of a robot to tf2. Once the state gets published, it is available to all components in the system that also use tf2. ROS Wiki

slam_toolbox (SLAM)

Simultaneous localization and mapping (SLAM) is the computational problem of constructing or updating a map of an unknown environment while simultaneously keeping track of an agent's location within it. There are several algorithms known for solving it, at least approximately, in tractable time for certain environments. Popular approximate solution methods include the particle filter, extended Kalman filter, Covariance intersection, and GraphSLAM. Wikipedia. There are several existing open-source laser scanner SLAM algorithm ROS packages for performing SLAM available for use, this include GMapping, Karto, Cartographer, and Hector. The reason for picking slam_toolbox over all of these options is the fact that slam_toolbox is more robust, most modern and has a strong backing by the ROS community. This package, slam_toolbox is open-

source under an LGPLv2.1 at SteveMacenski and is available in every current ROS distribution. It was also selected as the new default SLAM vendor in ROS 2, the second generation of robot operating systems, replacing GMapping. SLAM Toolbox was integrated into the new ROS 2 Navigation2 project, providing real-time positioning in dynamic environments for autonomous navigation (Macenski, Martín, et al., 2020). It has been shown to map spaces as large as 24,000 m2 The Journal of Open Source Software This package subscribe to `sensor_msgs/LaserScan` in the `/laser_scan` topic and tf message in the`/tf` and the uses the message to generate a `nav_msgs/OccupancyGrid` and publishes it to the `/map` topic message. It also helps the robot Localize itself with the environment.

RViz (Data Visualization)

This is a ROS inbuilt GUI tool for visualizing all the types of ROS messages published through ROS. The component is used to view the `nav_msgs/OccupancyGrid`message publised through the `/map` topic, the robot model, the odometry of the robot and much more. It is used to viualize all the ros messages publish within the Architecture. ROS Wiki

MoveBase (Dijkstra's Navigation)

The move_base package provides an implementation of an action (see the actionlib package) that, given a goal in the world, will attempt to reach it with a mobile base. The move_base node links together a global and local planner to accomplish its global navigation task. It supports any global planner adhering to the `nav_core::BaseGlobalPlanner` interface specified in the nav_core package and any local planner adhering to the `nav_core::BaseLocalPlanner` interface specified in the nav_core package. The move_base node also maintains two costmaps, one for the global planner, and one for a local planner (see the costmap_2d package) that are used to accomplish navigation tasks. ROS Wiki. The package is the package that is used to accomplish the navigation task, a goal of a target pose is publised from the Unity Component and the move_base server sends the required `Twist` message to the `cmd_vel` topic to move the robot.

> For each of the external packages, the parameters of the package have been tuned specifically for use on this robot and various launch files have been created to easily launch the required parts of the packages.

MapBased Navigation

This architecture uses the already created map from the SLAM architecture to navigate the known environment. For this architecture, the SLAM package is no longer required because the map is already known and another package can be used to Localize the robot on the map with Monte Carlo Localization. The Unity Component remains thesame as that of SLAM navigation. A detailed description of the two added component is attached below.

Map Server

map_server provides the map_server ROS Node, which offers map data as a ROS Service. It also provides the map_saver command-line utility, which allows dynamically generated maps to be saved to file. ROS Wiki. The map_saver node is the node that is used to save the map that is generated from the SLAM package. The

the map is saved as a .pgm and .yaml file the .pgm file is the image of the map and in the .yaml file, some information about the map like origin, path to map image etc is stored here. The map_server node is what would then be quired when the map data is required by amcl, rviz etc...

AMCL

amcl is a probabilistic localization system for a robot moving in 2D. It implements the adaptive (or KLD-sampling) Monte Carlo localization approach (as described by Dieter Fox), which uses a particle filter to track the pose of a robot against a known map. ROS Wiki. The package request for the map data from the map_server node and uses this data to localize the robot in the environment

Installation

> All the external ROS packages included in this project are ment for ROS Noetic Ninjemys, therefore the installation should be done on a machine with ROS Noetic Distro.

The project is intended to be done with two pc's, one Windows pc running Unity and an Ubuntu Pc running ROS, however there are ways around these to make the both of them run on one pc. Also the two pc's must be connected to thesame internet.

- Ubuntu Virtual Machine: Install an ubuntu virtual machine and then on the virtual machine install ROS and follow the steps bellow to install all the ROS packages required. **Note**; for the network adapter the bridge network option has to be selected.
- Docker Container: To use a docker container to run the ubuntu side of the project, a tutorial provided by Marco Gabriele Fedozzi can be followed by clicking here and then follow the instructions bellow for the installation of the ROS package
- A final option is to use **wslg** (*Windows Subsystem for Linux GUI*). Follow the installation instruction by clicking here and then follow the instructions below for the installation of the ROS package. *keep in mind that Windows 11 is required to run wslg*

UNITY INSTALLATION (WINDOWS)

- Visit https://unity3d.com/get-unity/download and download Unity Hub Through the Hub, install Unity 2020.2.2 (the version for which the projects have been developed) or later. Beware that later versions (eg. Unity 2021.1.x) may be incompatible with the projects ROS VERSION USED FOR THE PROJECT: ROS Noetic

- Visit https://github.com/TheEngineRoom-UniGe/SofAR-Mobile-Robot-Navigation to download the Unity project folder, extract it, then open Unity Hub and ADD the project to your projects list using the associated button.

- Open the Ubuntu system and enter the command below to retrieve the IP_Address of the Ubuntu system

    - ```
      ipconfig
      ```

    - copy the ip address from the **inet** section
      ```

- Open the Unity project In the bar on top of the screen, open the 'Robotics/ROS settings' tab and replace the 'ROS IP Address' with the IP of the machine running ROS that was just copied.

ROS INSTALLATION (UBUNTU)

First you create a folder for your catkin workspace

```
mkdir -p ~/sofar_ws/src
```

Clone the ROS Package repositories

for mobile_robot_navigation_project package

```
cd ~/sofar_ws/src
git clone --recurse-submodules https://github.com/Omotoye/Mobile-Robot-
Navigation-and-Mapping.git     # this will cause it to clone the submodule
together with the main repo.
```

for slam-toolbox package

```
cd ~/sofar_ws/src
git clone https://github.com/SteveMacenski/slam_toolbox.git
cd slam_toolbox
git checkout noetic-devel # to switch to the noetic branch
```

for installing all the ROS dependencies

```
cd ~/sofar_ws
rosdep install -q -y -r --from-paths src --ignore-src
```

Once the package has been successfully cloned, you then build the workspace

```
source /opt/ros/noetic/setup.bash
cd ~/sofar_ws/
catkin_make
```