UNIVERSITÀ DEGLI STUDI DI GENOVA

Software Architecture Project

Project Members

Name	Email Address
Omotoye Shamsudeen Adekoya	adekoyaomotoye@gmail.com
Farzin Sarvari	Sarvarifarzin94@gmail.com
Amanzhol Raisov	cornytravel@gmail.com
Taha Hussain Raja	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*
3. Bug0 Algorithm Navigation

Reactive Navigation (SLAM)

Reactive Navigation (SLAM) Software Architecture UML Component Diagram

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

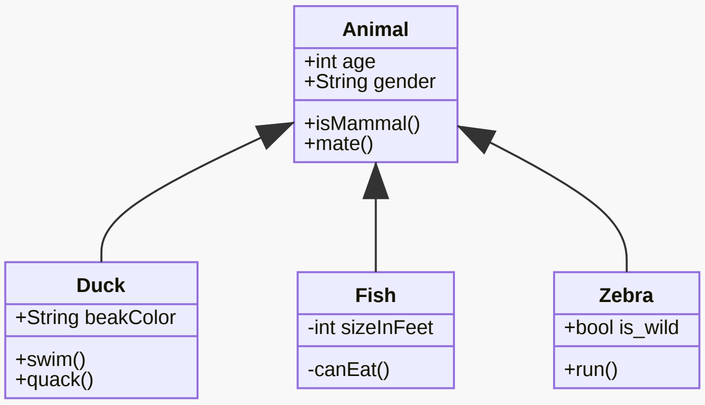
The is another ROS inbuild package that helps to publish the state of the robot to `/tf`. This package subscribes to joint states of the robot from the joint state publisher and publishes the 3D pose of each link using the kinematic representation from the URDF model. It publishes the state of a robot 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)

Syntax error in graph
mermaid version 8.11.5



Syntax error in graph
mermaid version 8.11.5