MR ROBOT

Surveillance Bot

PATH PLANNING

Robotics Assignment

Suliman S., Gaogle A., and Soma K.

2023

Group Members

Salmaan Suliman (2307945) | Aashna Gaogle (2202103) | Khiyara Soma (2003405)

Overview

This project was aimed at developing an autonomous navigation system for a TurtleBot using Gazebo for simulation, gmapping for building maps, and a custom node for navigation. The system was designed to explore an environment, identify a specific object (a green utility cart), and navigate to specific coordinates.

The primary goal was to design and implement a motion planning algorithm that could accurately navigate a robot in an environment simulated in Gazebo. The robot had to explore and map a specific section of the environment, detect the utility cart, and navigate to any given (x, y) coordinate provided.

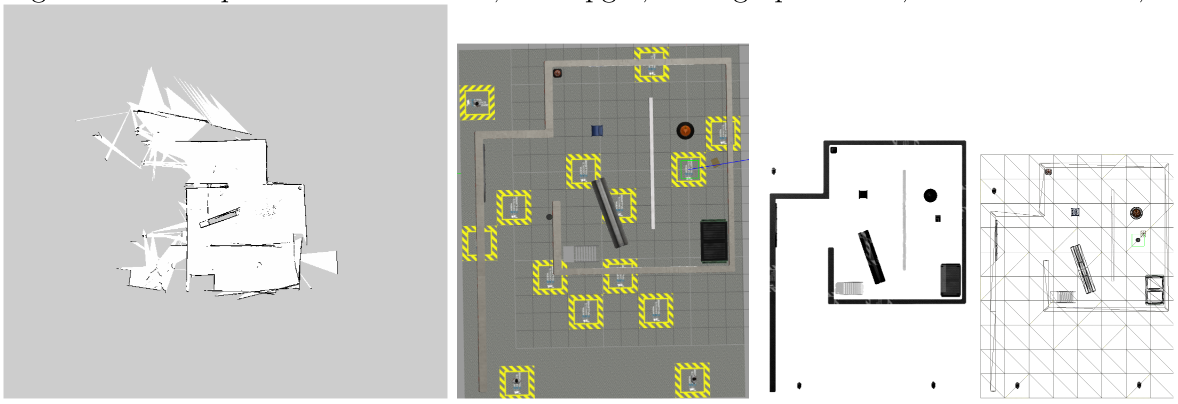
Mapping

The robot explored the environment using the gmapping package, a ROS package implementing SLAM (Simultaneous Localization and Mapping), was used to produce a 2D occupancy grid map. This was done in real-time as the robot moved around the environment, constantly updating its knowledge about its surroundings. The optimal performance of gmapping was ensured by tweaking its parameters, specifically setting the minimumScore argument to 100000.

Upon completion of the exploration, the map was saved, generating a .yaml file (containing metadata about the map) and a .pgm file (an image file representing the map itself).

The .pgm image was read and processed using OpenCV to convert it into an occupancy map. This map was then further processed to generate a binary image where the free and occupied spaces in the environment were represented by 0s and 1s, respectively.

Figure: The map in various formats; as a pgm, orthographic view, black and white, wireframe



Algorithm and Techniques

A PID (Proportional-Integral-Derivative) controller was utilised for the robot's movement control. The PID controller ensured that the robot navigated smoothly and accurately to the specified coordinates while minimising error.

The Rapidly-exploring Random Tree (RRT) algorithm was used for motion planning. This algorithm efficiently handles path planning in complex and dynamic environments, providing feasible paths from the robot's current location to the target coordinates.

The Gazebo state service was employed to obtain the robot's real-time location in the simulated environment.

The utility cart detection was implemented as a node that performed visual scans of the robot's view. Given that the cart is a large, bright green object, colour-based detection was used. If the cart was detected, a message was published to the '/witsdetector' topic.

The cart detection algorithm involved processing the robot's RGB images to isolate the green colour, which represented the utility cart. The robot then verified the size and shape of the detected green object to confirm it as the utility cart

Important Terminal Commands

References

The following references were used for assistance:

Kobuki: http://wiki.ros.org/kobuki/Tutorials/Examine%20Kobuki

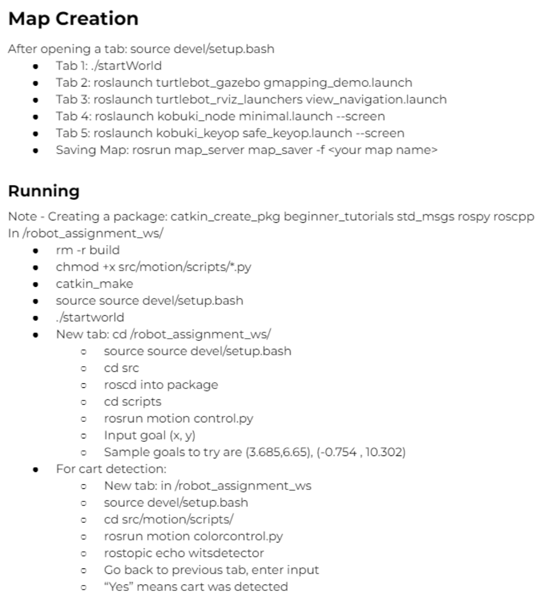
Navigation: http://wiki.ros.org/turtlebot_navigation/Tutorials/Autonomously%20navigate%20in%20a%20known%20map

TurtleBot: http://wiki.ros.org/turtlesim/Tutorials/Go%20to%20Goal

Package: http://wiki.ros.org/ROS/Tutorials/CreatingPackage

Mapping: http://wiki.ros.org/turtlebot_gazebo/Tutorials/indigo/Make%20a%20map%20and%20navigate%20with%20it

Customisation: http://wiki.ros.org/turtlebot/Tutorials/indigo/Customising%20the%20Turtle



Map Creation

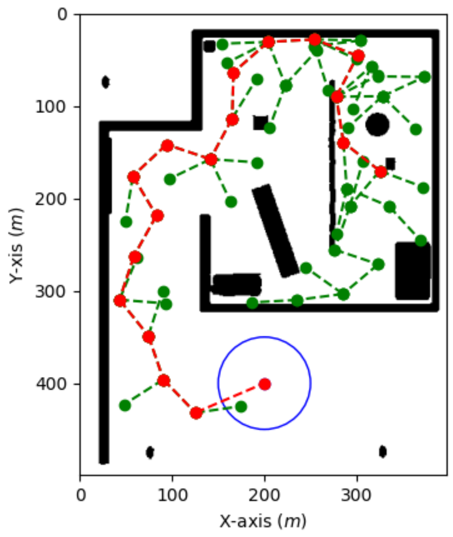
After opening a tab: source devel/setup.bash

- Tab 1: ./startWorld
- Tab 2: roslaunch turtlebot_gazebo gmapping_demo.launch
- Tab 3: roslaunch turtlebot_rviz_launchers view_navigation.launch
- Tab 4: roslaunch kobuki_node minimal.launch --screen
- Tab 5: roslaunch kobuki_keyop safe_keyop.launch --screen
- Saving Map: rosrun map_server map_saver -f <your map name>

Running

Note - Creating a package: catkin_create_pkg beginner_tutorials std_msgs rospy roscpp
In /robot_assignment_ws/

- rm -r build
- chmod +x src/motion/scripts/*.py
- catkin_make
- source source devel/setup.bash
- ./startworld
- New tab: cd /robot_assignment_ws/
  - source source devel/setup.bash
  - cd src
  - roscd into package
  - cd scripts
  - rosrun motion control.py
  - Input goal (x, y)
  - Sample goals to try are (3.685,6.65), (-0.754 , 10.302)
- For cart detection:
  - New tab: in /robot_assignment_ws
  - source devel/setup.bash
  - cd src/motion/scripts/
  - rosrun motion colorcontrol.py
  - rostopic echo witsdetector
  - Go back to previous tab, enter input
  - “Yes” means cart was detected


Y-xis (m)
X-axis (m)