MapMyWorld - Robotic Simultaneous Localization and Mapping

Christopher Leith

Abstract—In this project we create a simulated robot and demonstrate Simultaneous Localization and Mapping (SLAM) within the ROS framework to permit navigation within a small simulated "Gazebo" world. A differential drive robot is designed to operate within the Gazebo simulation and uses the ROS navigation stack and the RTAB-map-ros package for creating a map on the fly and localizing itself within that map.

Index Terms—Mobile Robot, SLAM, ROS.

1 Introduction

MOBILE robots must be able to determine their location within their region of operation, a process known as localization, and navigate to other locations reliably in order to perform their tasks succesfully.

If there is no predefined map of the region the robot must be able to build up a map of using its' sensors and simultaneously localize itself within that map. This process is known as Simultaneous Localization and Mapping (SLAM).

In this project SLAM is demonstrated in a supplied Gazebo world and a custom designed world.

2 Background / Formulation

Because mapping and localization is so important for mobile robotics it is a highly researched field and has many well developed techniques and technologies. The SLAM approach has proven a very effective solution for this problem and has developed into a family of similar technologies including Fast-SLAM, Graph-Slam and occupancy grid mapping each with with pros and cons for different applications.

2.1 Occupancy Grid

Occupancy grid mapping is a technique that uses bayes filtering to determine the presence of obstacles within a 2D grid of the environment.

2.2 FastSlam

FastSlam uses particle filtering. It uses comparisons of kalman filtered random particle poses against the current, known sensor data input at each moment.

2.3 GraphSlam

GraphSlam solves the Slam problem by analyzing the entire trajectory of robot poses, all of the previous sensor inputs and the current map. It attempts to correlate the sensor inputs into correlated features that then feedback into correcting the presumably noisy pose locations. This is the technique used by by RTABMap.

2.4 RTABMap Slam

In this project we demonstrate a mature SLAM framework implementation known as Real-Time Appearance-Based Mapping (RTAB-Map) available for 'plug-and-play' use in the ROS robotic framework. RTABMap is a graph-based SLAM implementation intended for robots operating for long periods in large environments. Internally it uses many techniques to efficiently accumulate large maps in real time even on the constrained computing resources typically available on mobile robotic hardware. In this project the RTABMap uses 3 sensor streams:

- Wheel Odometry Measures the distance traveled be each wheel.
- Hokuyo Laser Rangefinder Mounted on top of the cylinder.
- Kinect RGB-D Camera Mounted on the front of the cylinder.

The RTABMap package uses all of these sensor data streams in a way that is, frankly, somewhat opaque. It is primarily the RGB-D point cloud data that is used by the RTABMap package to detect visual features over the course of mapping whose geometric similarities correspond closely enough to infer that they must be images of the same feature. The detection of such a correspondence is called a 'loop closure'. As loop closures are determined they can be used reconcile the locations of seemingly disparate stored image features, as determined by odometry, into identical features of the map and thereby correct path inaccuracies due to noise of the odometry. Whether a loop closure is determined or not the new images are stored for later camparisons. As the dictionary of stored image features grows the processing load increases. RTABMap implements several data caching strategies to ensure that real time processing can continue.

3 Configuration

Config

3.1 Robot Config

The robot used in this project, Ciel-Bot, is an enhanced version of the very simple robot used in the previous localization project. It is a cylindindrical, 2 wheeled differential

drive robot with 3 sensors mentioned above. The transform tree is shown in Fig 1.

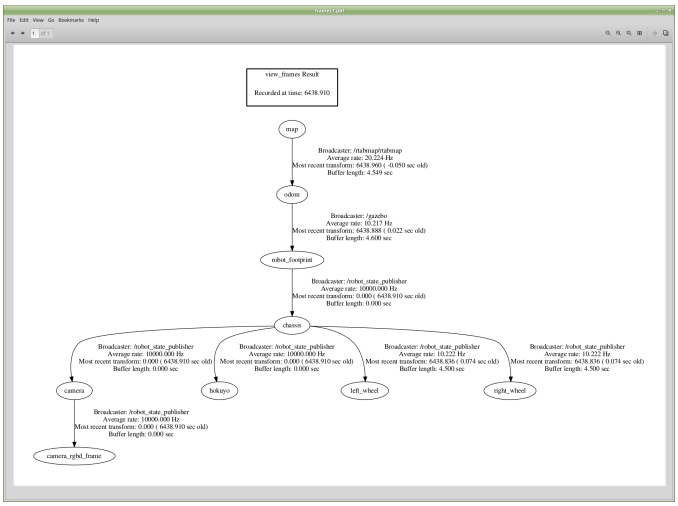

Fig. 1. CielBot Transform Tree

3.2 Custom World Config

4 RESULTS

Results

5 DISCUSSION

Discussion

6 CONCLUSION / FUTURE WORK

Conclusion