[PROJECT NAME]

Tanay Nistala, Stephanie Wilson, Iris Yang

Introduction

Background & Related Work

Approach

For a robot to traverse a maze (or any environment, for that matter), it typically requires a map of said environment either as a physical map or some form of costmap, which can then be used for pathfinding. However, in this scenario no such map is made available to the robot.

Producing a map of the environment is possible with the right sensor data and odometry, and simultaneous localization and mapping (SLAM) algorithms provide a means of doing so while also allowing the robot to localize itself within the unfamiliar environment. A similar issue is raised here, however, as these algorithms typically require manual teleoperated control through the unknown environment in order to build the map for future use.

This creates a dilemma: traversing an environment autonomously requires a previously prepared map, which can itself only be produced through manual navigation. However, the modular framework provided by the Robot Operating System (ROS) lends itself well to the approach devised for solving this problem. For [PROJECT NAME], separate nodes can be initialized for navigation and SLAM, wherein the former "disguises" itself as a manual controller for the robot, while using odometry data published by the latter concurrently.

Further sections explain the setup of such a system in greater detail.

Sensor Data & Odometry

Sensor data and odometry are crucial for the localization of the robot, as this forms the base of reliability for future steps of [PROJECT NAME]. Robust localization leads to more trustworthy data for mapping within the SLAM node, which in turn increases the accuracy of odometry passed on to the navigation node. Two main data sources are used for this: raw positional odometry from the robot's drive base, and LIDAR scanner measurements.

The robot's drive base continually computes estimates the current position and pose (heading) from a calibrated base point (which is typically the point of initialization). This utilizes known data about the size of the robot's wheels along with the drive motors' operating speeds:

$$\mathbf{v} = \begin{bmatrix} v_l \\ v_r \end{bmatrix}$$
$$\mathbf{w} = \frac{1}{r_{wheel}}\mathbf{v}$$
$$\mathbf{w}_{wheel} = \begin{bmatrix} w_l \\ w_r \end{bmatrix} = \delta_t \mathbf{w} = \frac{\delta_t}{r_{wheel}}\mathbf{v}$$
$$\delta_s = r_{wheel}\frac{w_l + w_r}{2}$$
$$\delta_\theta = r_{wheel}\frac{w_r - w_l}{\delta_{wheel}}$$

where δ_{wheel} is the separation between the robot's wheels.

LIDAR scans provides distance estimates to nearby obstacles in the environment. For simplicity, this is implemented as three laser beams: one directly ahead (0°), and two offset to the left and right (30° and -30°-330°). Since the LIDAR scanner rotates as it publishes data, this produces a 360° map of the environment from the robot's position; the main caveat here is that LIDAR sensors have a maximum range of measurements, resulting in limited data in large environments.

Navigation

Navigation is achieved through the dynamic window approach (DWA) method developed by Fox et al. (1997), and is provided wholesale as part of TurtleBot's navigation libraries. This algorithm plans paths based on the available world map; for known areas of the map it can plan paths that avoid walls and obstacles between the robot and the destination, but plans a direct path to the destination in unknown areas of the map, lacking any data about the environment there. This algorithm takes into account the robot's own motion dynamics, such as turn radius and speed/acceleration. This allows for far more accurate path planning, and results in fewer instances of the robot getting itself stuck in tight spaces.

Simultaneous Localization & Mapping (SLAM)

SLAM algorithms bridge the gap between odometry and the navigation layer of the system, providing a means for interpreting the former to create a map for the latter. Odometry and LIDAR scans published by their respective nodes are consumed by the SLAM algorithm, which uses prior data about the environment and the robot's pose to localize the robot and potentially expand the internally generated map. In turn, the robot's (estimated) location and the map are published by the SLAM node for use by the navigation layer described above.

SLAM itself faces a similar problem experienced here with regards to the interdependence of multiple variables. For localization, an accurate and consistent

map is required, but for mapping an accurate location is needed. Three main SLAM algorithms are utilized here, each using various sources of data: GMapping, Hector, and Karto.

GMapping, introduced by Grisetti et al. (2005, 2007), is one of the most accurate methods available for solving the SLAM problem, and utilizes a Rao-Blackwellized particle filter where each particle contains a separate map of the environment. Crucially, it relies on both robot odometry and LIDAR scans, which helps it achieve a high level of accuracy, but is still more tolerant of noise in the robot's motion model and sensors as compared to an Extended Kalman filter (EKF).

Hector was introduced by Kohlbrecher et al. (2011), and uses an EKF instead. Designed to be much faster to process, it relies heavily on LIDAR data as odometry is optional. Localization is achieved through comparisons with nearby features, but requires little noise in sensor data and motion estimates. As such, it is more tolerant of drops in the odometry data stream, but typically this comes with a tradeoff in that the robot's motion must remain slow and controlled to avoid losing reference points too quickly.

Karto, developed by Konolige et al. (2010), achieves a balance between the aforementioned methods using a method called Sparse Pose Adjustment (SPA) that optimizes pose graphs. Importantly, it is tolerant of failure of odometry, unlike Hector; when lacking odometry data it can still generate consistent SLAM poses.

These three methods do create generally accurate maps of the robot's environment, but here we aim to investigate the effect of the SLAM method used on the robot's ability to autonomously navigate a new environment in addition to mapping it.

Demonstration

Conclusion & Future Work

References