Docking for Autonomous Mobile Robots

Chris Rowe ‘20
Advised by Prof. Fixel
Trinity College Department of Engineering

Abstract

Docking is a specific utilization of landmark localization – comparing detected local features against a known map containing unique “landmark” features to determine current local coordinates. This capstone aims to develop a ROS package that adds general docking functionality to an autonomous mobile robot of differential drive configuration. The package must be capable of determining a valid pose-sequence that can mate the robot with the dock. The design and implementation of each of the stages within the perception pipeline are described, and the results obtained from running the perception pipeline are provided. Similarly, the design and implementation of the motion planner are described and analyzed, and results are provided.

Introduction

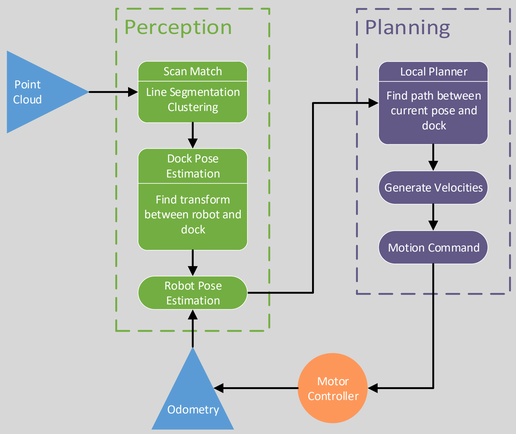
Localization: The process by which a robot determines its pose (representation of its position and orientation) within a parent coordinate frame.

Docking: A specific utilization of “landmark localization”, comparing detected local features against a known map containing unique “landmarks”. Any task given to a mobile robot requires it to move from its current pose to a pose. Thus, docking and localization in general are fundamental components of any system involved in robotics, and autonomous navigation.

Robot Operating System (ROS): Software framework created to standardize robotics development tools. Software are provided through modules called packages, each of which provides a unique functionality. Among the packages available through ROS, there are some that perform odometry, path planning, localization etc.

Design Approach

The docking software has two main modules: the perception pipeline and the motion planner. The perception pipeline processes the point cloud to extract the dock and estimate its pose relative to the robot. The motion planner uses the estimated pose to calculate the plan that the robot must follow to arrive at the dock pose.

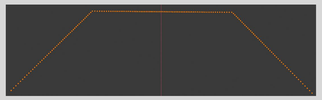


Overview of software structure

Dock Model Template

To identify the dock within an input point cloud, the CAD model of the dock must be supplied by the user. There is a tutorial provided for the user to follow in order to obtain a .PLY point cloud file from the CAD model. The .PLY point cloud will be used as a reference for matching the dock template to points within the input point cloud.

Top-down view of dock CAD model



Point cloud template obtained from CAD model

Dock Detection Perception Pipeline

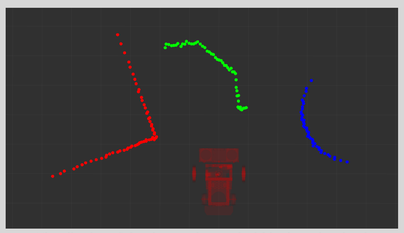
LIDAR SCAN

A raw point cloud is input from the onboard LiDAR, which takes a 2D scan of the surrounding environment. This point cloud contains the surfaces of all objects detected within the robot’s environment, including the dock itself and various obstacles that must be extracted.

LiDAR scan containing the dock along with environmental obstacles

CLUSTERING

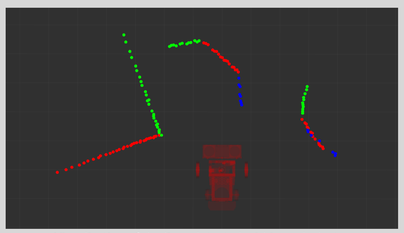
Density-based clustering is performed on the raw point cloud to separate the input cloud into individual clouds, each of which contains only points belonging to a single object. This reduces the computation load for further processes since each cluster can now be analyzed individually.



Visualization of the clusters detected within the point cloud

LINE DETECTION

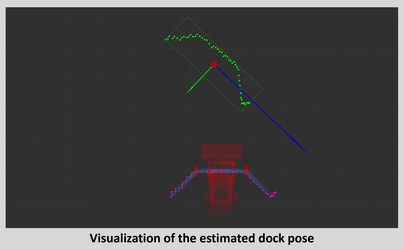
The Random Sample Consensus (RANSAC) algorithm is performed on each cluster cloud to identify any lines. In cases where the dock shape is made only of lines, clusters containing no lines can be ruled out as a potential dock cluster. Since the next step is computationally expensive, elimination of obstacle clusters reduces computation load.



Visualization of the lines detected within each cluster

POSE ESTIMATION

The Iterative Closest Point (ICP) algorithm is applied to each cluster marked as a potential dock cluster. ICP uses the reference point cloud obtained from the supplied dock model as a template and attempts to match the measured clusters to the template. ICP works by iteratively applying transformations containing translations and rotations to the input point cloud and calculates the mean squared distance error between corresponding features as a result of the applied transformation. The cluster with the smallest error is most likely the dock cluster, and the corresponding transformation is used as the pose transformation between the robot and the dock.



Visualization of the estimated dock pose

Motion Planning

When the dock’s pose relative to the robot is successfully identified, the motion planner calculates a plan composed of a sequence of velocity commands that enables the robot to move from its initial position and orientation to the target position and orientation of the dock.

The plan is calculated using a feedback control law [1], where the control error variables for radial distance (**r**), current heading, (**δ**) and target orientation (**φ**) are reduced to zero through the control variables for linear velocity (**v**) and angular velocity (**ω**).

The feedback control law has several variables which can be adjusted to modify the generated path based on physical properties of the robot and preferences of the user.

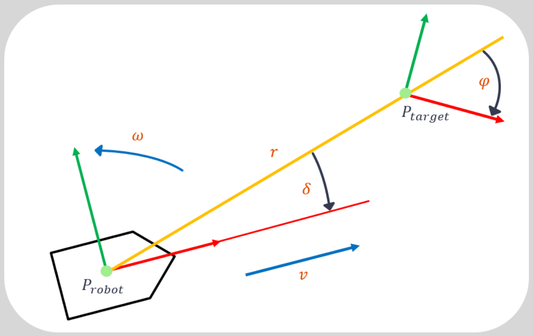
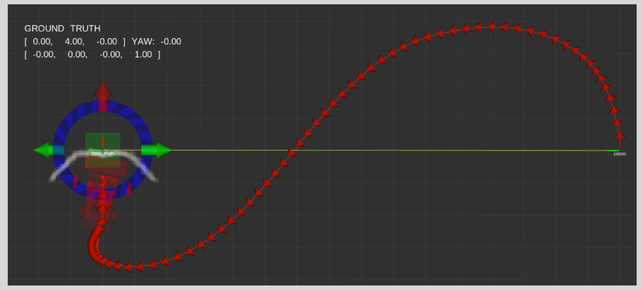


Diagram showing control system variables

Results

From a raw LiDAR point cloud, the program is able to use a supplied template of the dock to isolate the points belonging to the dock and extract them into a separate cluster.

The program is then able to successfully estimate the pose of the dock using ICP. In simulation, pose estimation can be accurate within 1-2cm depending on the distance resolution of the LiDAR simulation. Live testing was performed on the Turtlebot LDS-01 LiDAR, which has a distance accuracy of ± 1.5cm and achieved accuracy within 5-8cm.



Example simulation run of dock pose detection along with motion plan generation and execution.

With a supplied dock pose, the motion planner can successfully generate smooth trajectories that respect the dynamic constraints of the robot. Using the GUI, the gains can be adjusted to fit the curvature, velocity and acceleration preferences of the user. At the default planning rate of 10Hz, the planner will output successful trajectories. However, the lower the frequency, the lower the update rate, meaning the robot can veer off course longer without the planner readjusting.

The design report and ROS package can be downloaded from http://github.com/rwbot/docking.

Future Work

ROS Navigation Stack Integration: The ROS Nav Stack is a collection of packages supporting autonomous navigation. The Nav Stack is organized into modules having standardized communication and has multiple packages that can be chosen from when setting up a module. In its current state, the docking package functions only as a standalone package. Due to time constraints, the package has not yet been modularized into a component easily integrable with the nav stack. Simply, this means that a robot can either use the nav stack to navigate, or use the docking package to perform docking, but not both at the same time.

Obstacle Avoidance: The perception pipeline works under the assumption that there are no obstacles between the robot and the dock. This is a logical conclusion, since an obstacle between the LiDAR and the dock would prevent the LiDAR from scanning the dock. Similarly, the feedback control law used by the motion planner also shares this limitation, not only because it depends on an input dock pose from the perception pipeline in order to work, but also because fundamentally the control law does not consider obstacles. As such, for the robot to be of practical use, an obstacle avoidance would be required to handle navigation/docking in the presence of static and dynamic obstacles.

Acknowledgements

• Prof. Fixel
• Prof. Huang
• Nancy Fleming
• Prof. Mertens
• Prof. Byers
• Andrew Musulin

References

[1] Park, Jong Jin. “Graceful Navigation for Mobile Robots in Dynamic and Uncertain Environments.” (2016). https://deepblue.lib.umich.edu/handle/2027.42/120760