Navigation, Exploration and Search Methods for Residential Fire Fighting Robot

Introduction

Smart home systems serve for more than home automation, they can be employed to protect the homeowner's family and investments. A residential fire fighting robot may be the smart home's next great companion due to the decreased response time to a fire it could provide. Today robots are deployed in various industries for operation in situations which are too dangerous or inaccessible for humans. These functionalities have been made possible by advancements in robotics communication frameworks and perception, navigation, and exploration algorithms that allow a robot to correctly interact with its environment. This technical paper reviews currently available robotics open-source frameworks, libraries, and methodologies used to implement ROS software packages for navigation and obstacle avoidance, map exploration, and object search. All after-mentioned software is open source and free to download, modify, and use in accordance to the standard BSD license.

Robotic System Development Methods

The construction of a complex robotic system does not need to start completely from scratch as there are open-source robotics frameworks readily available for consumption. A robotics framework is a collection of software tools, libraries and conventions, aiming at simplifying the task of developing software for a complex robotic device. A robotics framework should also contain a middleware which is similar to firmware as it dictates the communication network between low-level hardware and software of the robot. The Robot Operating System (ROS) is a linux-based framework agreed to be the state-of-the-art of robotics frameworks [1] as it provides a low-level communications infrastructure, a open-source robotics toolset for perception and navigation that is being continuously developed by the robotics community, and a powerful development toolset for debugging and visualization.

ROS has gaps in functionalities that other robotics frameworks address such as simulation, real-time controls infrastructure, and networking with Internet of Things (IoT) devices; however due to its popularity, ROS allows seamless integration with other robotics libraries. For a smart home residencial robot networking with other embedded devices over the internet is required. The Hop robotics framework (a.k.a RAPP) is a client-server programming environment built on top of web protocols in Javascript as a toolset for programming the "Web of Things" with a cloud computing infrastructure. Hop integrates with ROS and can be used to extend home automation to ROS-incompatible 3rd party hardware [1] like smoke detectors. Development of an artificial intelligence for search and object detection will require a simulation tool like Gazebo 3D which integrates with ROS to allow full environment simulation with support for four different physics engines including DART, the Dynamic Animation and Robotics Toolkit from Georgia Tech. DART is unique in its ability to give full access to internal kinematics of objects in simulation [2] and is useful for simulating the control of robotic arms. Real-time control is required to aim and operate the fire fighting robot's fire extinguisher. Digesting a continuous signal of data needed for implementing a feedback control system in ROS is made possible by Orocos RTT which creates thread-safe and efficient ports in ROS's network infrastructure for lock-free data exchange [3].

Indoor Navigation and Obstacle Avoidance

To further extend the fire fighting robot's problem description, the robot's available sensors for perceiving the world include a Hokuyo 2D lidar, ultrasonic distance sensors, and a monocular camera. Navigation of a known map is preferred over performing simultaneous localization and mapping (SLAM) in an unknown environment since presence of smoke degenerates the accuracy of lidar data [5]. An initial map, such as a home floor plan, can be imported into the map_server node of the ROS Navigation Stack. Individual static objects can be integrated into the map using the transformation node TF which can help reduce localization error and prevent the robot from getting lost by acting as landmarks [6]. ROS navigation stack also has two basic recovery processes for when the robot is stuck or lost which can be further enhanced by creating a finite state machine specific to the application by using the SMACH python library [4].

Navigation of a robot involves maintaining a map of the environment, localizing the robot to the map, and planning the robot's path to its destination and around obstacles. The ROS Navigation Stack consists of nodes, processes that take in sensor streams or messages from other nodes, that work together to navigate the robot to waypoints by sending velocity commands to the robot's move_base node. The components that plan the robots path through the map and around obstacles are the Global Planner and the Local Planner which use the Dynamic Window Approach (DWA) algorithm to break the global trajectory into small trajectories for simulating paths around obstacles, predicting their results, and scoring the trajectory of each to choose the best path. The global planner has different nodes the user can choose to perform path planning with a specific graph algorithm like Dijkstra's or A* [4]. The Navigation Stack also uses sensor streams to build local and global cost maps, which are occupancy maps and only the local cost map is generated from the robot's sensors in real time.

Map Exploration and Object Search

The map exploration problem and the object search problem have the same general structure, but their objectives are dissimilar for the trajectory that is optimal in exploration does not minimize the value of time to find an object along it [7]. Frontier based, feature based, hector exploration planner, and forward simulation exploration algorithms are available as ROS packages. The ROS frontier_exploration package integrates with the ROS navigation stack to define a boundary for exploration, break the grid of the map into "frontiers" and then uses the greedy algorithm approach to visit all frontiers of the map, starting with the closest frontier which may be inefficient because the utility or value of exploring that frontier may be less than other readily available frontiers. In feature-based exploration the robot is pushed toward new feature discovery in the map instead of creating grid-based frontiers which is less computationally expensive; however the features that successfully augment the map depend on the task of the robot and unless a certain area offers several points of interest for the robot and increases the utility of traveling to that area by offering new information, the robot will not go there. Hector Exploration Planner algorithm is developed on top of the ROS frontier exploration package, but unlike frontier exploration, hector planning does not choose the next frontier with the closest distance but instead calculates the distance to the closest obstacles and chooses the frontier that has a path that is most likely reachable [8]. The forward simulation algorithm is also developed on top of the ROS frontier exploration package, but unlike hector planning forward simulation uses a Markov Decision Model, which looks at all possible next states and chooses the best option the will

maximize long term reward, to choose the next frontier for exploration [2]. Hector planning has shown many promising results in rescue missions and may be the best option for developing a fire searching and extinguishing robot.

[1] E. Tsardoulias and P. A. Mitkas, "Robotic frameworks, architectures and middleware," 2017. [Online]. Available: https://www.researchgate.net/publication/321180717_Robotic_frameworks_architectures_and_middleware_comparison. [Accessed: March 1, 2019].

[2] M. Lauri, R. Ritala, "Planning for robotic exploration based on forward simulation," 2016. [Online]. Available: https://arxiv.org/pdf/1502.02474.pdf. [Accessed: March 7, 2019].

[3] Orocos Real-Time Toolkit, "The Orocos Real-Time Toolkit," [Online]. Available: http://www.orocos.org/rtt. [Accessed: March 3, 2019].

[4] K. Zheng, "ROS Navigation Tuning Guide," 2016. [Online]. Available: https://arxiv.org/pdf/1706.09068.pdf. [Accessed: March 3, 2019].

[5] J. W. Starr, B. Y. Lattimer, "A Comparison of IR Stereo Vision and LIDAR for use in Fire Environments," 2013. [Online]. Available: https://ieeexplore.ieee.org/document/6411591. [Accessed: March 5, 2019].

[6] R. Mata, "Persistent Autonomous Exploration, Mapping and Localization," 2017. [Online]. Available: https://dspace.mit.edu/bitstream/handle/1721.1/113127/1017567128-MIT.pdf?sequence=1. [Accessed: March 7, 2019].

[7] M. Kulich, L. Preucil, J. Bront, "Single Robot Search for a Stationary Object in an Unknown Environment," 2014. [Online]. Available: https://ieeexplore.ieee.org/document/6907716. [Accessed: March 7, 2019].

[8] S. Wirth, J. Pellenz, "Exploration Transform: A stable exploring algorithm for robots in rescue environments," 2013. [Online]. Available: https://www.uni-koblenz.de/~agas/Documents/Wirth2007ETA1.pdf. [Accessed: March 7, 2019].