Flexible Navigation: Finite State Machine-Based Integrated Navigation and Control for ROS Enabled Robots

David C. Conner and Justin Willis
Capable Humanitarian Robotics and Intelligent Systems Lab (CHRISLab)
Department of Physics, Computer Science and Engineering
Christopher Newport University
Newport News, VA 23606
Email: {david.conner , justin.willis.13}@cnu.edu

***Abstract*—This paper describes the *Flexible Navigation* system that extends the ROS Navigation stack and compatible libraries to separate computation from decision making, and integrates the system with FlexBE — the Flexible Behavior Engine, which provides intuitive supervision with adjustable autonomy. Although the ROS Navigation plugin model offers some customization, many decisions are internal to move_base. In contrast, the *Flexible Navigation* system separates global planning from local planning and control, and uses a hierarchical finite state machine to coordinate behaviors. The *Flexible Navigation* system includes Python-based state implementations and ROS nodes derived from the move_base plugin model to provide compatibility with existing libraries as well as future extensibility. The paper concludes with complete system demonstrations in both simulation and hardware using the iRobot Create and Kobuki-based Turtlebot running under ROS Kinetic. The system supports multiple independent robots.**

I. Introduction

ROS[1], the Robot Operating System, is widely used in everything from student research to advanced humanoid robotics used in the DARPA Robotics Challenge[1]–[3]. One widely used ROS *package* is ROS Navigation[2], shown in Figure 2, which combines 2D occupancy mapping with global and local planning to move a robot while detecting and avoiding obstacles[4]. This tightly coupled package works well, but is rigid in its operation. In contrast, one of the lessons learned from the DARPA Robotics Challenge was the benefit of *collaborative autonomy*, and having systems with the flexibility to allow operators to interact with an autonomous system as needed[3].

This paper builds upon the recently open sourced Flexible Behavior Engine[3] (FlexBE), and develops an extension of the ROS Navigation system that permits supervision and collaborative autonomy using hierarchical finite state machines. The *Flexible Navigation* system, which is available open source[4], allows the state machine designer to allow operator approval of global plans prior to execution on the robot, to interrupt execution at any time and re-plan as needed, and to develop custom recovery behaviors that can be invoked in particular situations. FlexBE allows these behaviors to be modified during runtime if needed[5]. The design of Flexible Navigation allows existing planning and cost map plugins that are compatible with ROS Navigation to be used within the flexible framework.

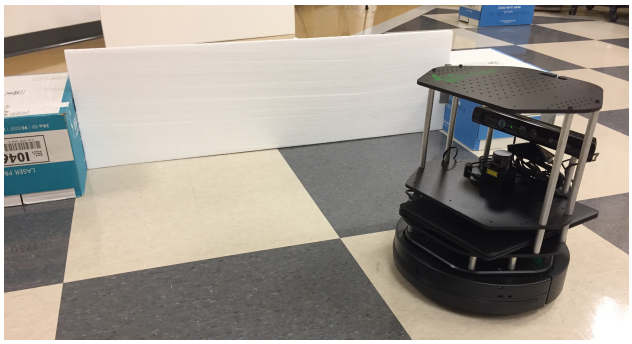


Fig. 1. Turtlebot using Flexible Navigation to move around a set of obstacles.

Section II places this work in the larger research context, and provides an overview of existing ROS-based navigation capabilities. Section III provides an overview of the Flexible Behavior Engine (FlexBE). Section IV describes the contribution of this paper, which uses FlexBE to extend the capabilites of the ROS Navigation system to enable collaborative autonomy in mobile robot navigation. Section V describes the experimental validation in both simulation and hardware using a Turtlebot[5] mobile robot as shown in Figure 1. The paper concludes with an overview and future work discussion.

II. Related Work

Most modern robotic systems use hybrid architectures that combine higher update rate controls and behavioral subsystems with with lower-rate planning and perception sub-systems;

[1]http://www.ros.org
[2]http://wiki.ros.org/navigation?distro=kinetic
[3]http://wiki.ros.org/flexbe
[4]http://wiki.ros.org/flexible_navigation
[5]http://www.turtlebot.com

978-1-5386-1539-3/17/$31.00

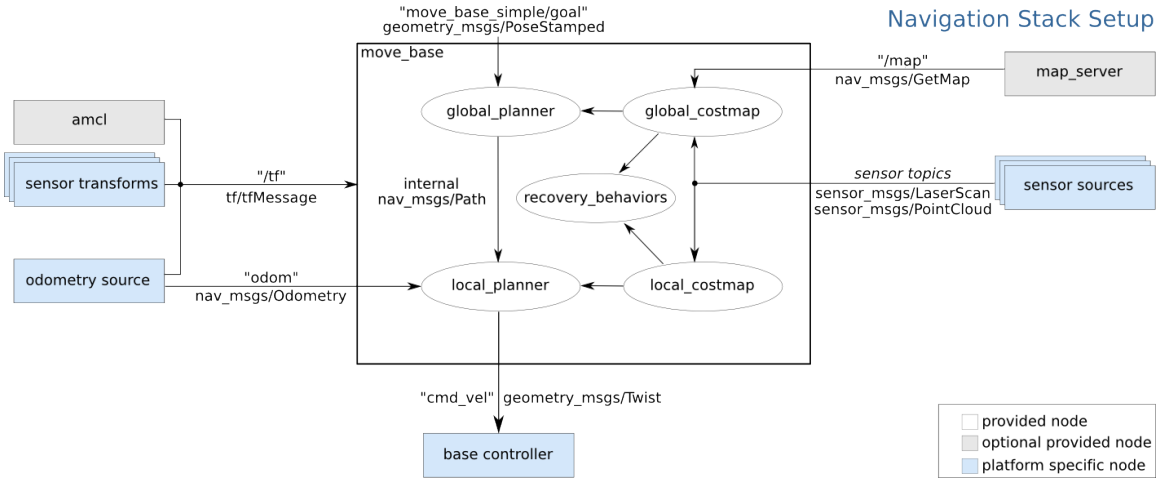


Fig. 2. Basic architecture of ROS Navigation Stack using localization to a known map. (credit http://wiki.ros.org/move_base?distro=kinetic)

these hybrid deliberative/reactive systems represent a fusion of fast reacting behavioral systems with deliberative planning systems[6]. Such hybrid systems were used to great success in the recent DARPA Challenges, including the second DARPA Grand Challenge[7], the DARPA Urban Challenge[8]–[10], and the recent DARPA Robotics Challenge[1]–[3], [11]–[13]. Eighteen of the 23 teams participating in DARPA Robotics Challenge (DRC) based their hybrid architectures around the open source Robot Operating System (ROS)[14].

A. Robot Operating System (ROS)

The ROS software system began development under the STandford AI Robot (STAIR) project, before being incubated at Willow Garage beginning in 2007[15], [16]. Since its introduction as an open source project, ROS has enjoyed exponential growth, and has been adopted by robotics researchers around the world[17]. In recent years, the Open Source Robotics Foundation[6] (OSRF) has taken over management of the ROS infrasture. Through OSRF, DARPA invested significant resources into the upgrade of the Gazebo simulation framework that is integrated with ROS[18]. By OSRF's count, 18 of the 23 teams in the DRC Finals used ROS, and 14 teams used Gazebo as their primary simulation framework[14]. With the recent publication of multiple text books[16], [19], [20], ROS seems poised to continue its reign as the "de facto standard in robotics".

ROS provides peer-to-peer communication middleware, tools for building and deploying complex robotic software systems, and a rich library of components, all within a multilingual (e.g. C++, Python, ...) environment[16]. These *packages*, as they are called in the ROS ecosystem, fit well within the standard modular hybrid deliberative/reactive frameworks. Common packages include ROS Control[7] for low-level system control and interface, common sensor drivers such as the Hokuyo LIDAR[8] and Microsoft Kinect[9], and perception frameworks such as OpenCV[10] and the 3D Point Cloud Library (PCL)[11][21]. Several cutting edge planning frameworks have ROS wrappers, including the Open Motion Planning Library[12] (OMPL) and the Search Based Planning Library[13] (SBPL)[22], [23]. MoveIt![14] integrates these planning systems with 3D perception and robot modeling to provide manipulation and motion planning for high degree of freedom systems such as robot arms[3], [24]. In addition to robot system, ROS provides tools for visualization (RViz[15]) and data logging and replay[16].

B. Localization, Navigation, and Planning

One of the most widely used and mature family of packages is the ROS Navigation system [4], [25]–[28]. ROS Navigation, as shown in Figure 2, provides components to build 2D occupancy (cost) maps[17] from scan data (e.g. from LIDAR sensors) as shown in Figure 3, and perform either Adaptive (KLD-sampling) Monte Carlo Localization (AMCL)[18][29] to a known map, or Simultaneous Localization And Mapping (SLAM)[19][30]. Given the robot pose estimate, the *move_base* component performs coordinated two-layer planning using a global planner to find a path to a designated goal, and a local planner that attempts to follow the global path. The local planner sends velocity commands to the robot. The move_base component automatically detects when it is stuck,

[6]http://www.osrfoundation.org
[7]http://wiki.ros.org/ros_control
[8]http://wiki.ros.org/urg_node
[9]http://wiki.ros.org/openni_launch
[10]http://wiki.ros.org/vision_opencv
[11]http://wiki.ros.org/pcl
[12]http://ompl.kavrakilab.org
[13]http://sbpl.net/Software
[14]http://moveit.ros.org
[15]http://wiki.ros.org/rviz
[16]http://wiki.ros.org/rosbag
[17]http://wiki.ros.org/costmap_2d?distro=kinetic
[18]http://wiki.ros.org/amcl
[19]http://wiki.ros.org/gmapping

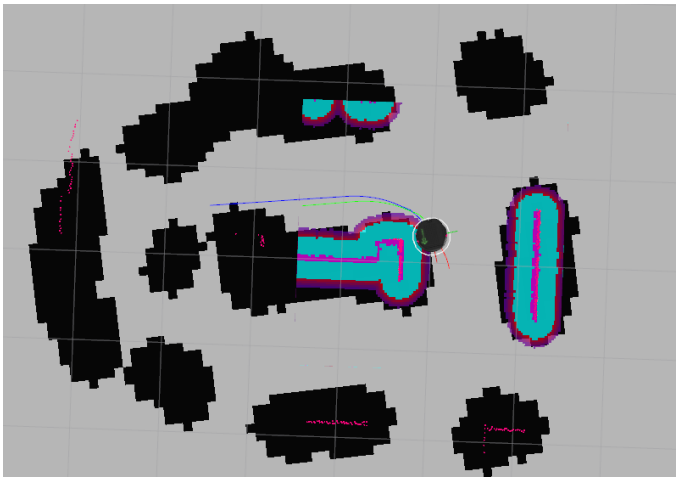


Fig. 3. RViz display of global and local costmaps created by ROS costmap_2d components during simulation.

and can initiate automatic recovery behaviors such as clearing the current cost map and rotating in place to refresh the map.

The move_base component uses a plugin model that accepts several different planning approaches including Dijkstra's algorithm, A*, and Navigation Function based planners that are provide with ROS Navigation[4]. A plugin wrapper[20] also exists for state lattice-based planners from the Search-based Planning Library[23], [31]. Recovery behaviors can also be customized via additional plugins. While this tightly integrated perception, planning, and control system works well in many cases [4], [25]–[28], it removes control from the user after a global path to a defined goal is found. This tight coupling prevents *collaborative autonomy*, and inspires the main contribution of this paper.

C. Hierarchical Finite State Machine-based Behaviors

As introduced above, many modern systems use a hybrid architecture for complex autonomous systems that use hierarchical finite state machines (HFSM) to define the overall system behavior and coordinate between sub-systems [3], [7]–[10]. In contrast to previous DARPA challenges that required complete autonomy, the DRC was conceived around the concept of human-robot teams that could interact and enable a better response to disaster scenarios[11]. In the DRC, the human operator/supervisors could augment the robots perception and decision making capabilities, which gave rise to the concept of *collaborative autonomy*[3].

Figure 4-a shows an example of a HFSM-based behavior used by Team ViGIR in the DARPA Robotics Challenge Finals. Here, the operators needed to be able to interrupt the robot if errors were detected, and pass data to the robot behavioral system as needed. Complicating this scenario were the artificially restricted communications used in the DRC.

ROS provides a Python-based system called SMACH[21] for constructing and managing HFSM[32]. While several teams used SMACH-based HFSM during the DRC[1], [33], SMACH is not designed for collaborative autonomy or monitoring under remote operation, which motivated Team ViGIR to extend SMACH and develop FlexBE – the Flexible Behavior Engine[5], [34].

[20]https://github.com/CNURobotics/sbpl_lattice_planner

[21]http://wiki.ros.org/smach

III. FlexBE — The Flexible Behavior Engine

FlexBE is a major extension of SMACH that provides a number of benefits[5], [34]. FlexBE provides an advanced graphical user interface (GUI) for constructing HFSM and monitoring their execution. FlexBE uses a state machine *mirror* concept that permits monitoring and control across a potentially sporadic communications pipeline. Figure 4-a shows the FlexBE Editor UI with an example turn valve behavior used during the DRC.

FlexBE extends SMACH's basic concept of a Python-based *state implementation* to include the concept of an adjustable *autonomy level* as shown in Figure 4-b. This allows the HFSM designer to specify a required level of autonomy for automatic transition between states, and the operator to adjust the current autonomy level based on changing conditions. The Python-based state implementations can acquire data using the standard ROS messaging interfaces, pass data between states as *user data*, or allow the operator to inject data via the FlexBE UI.

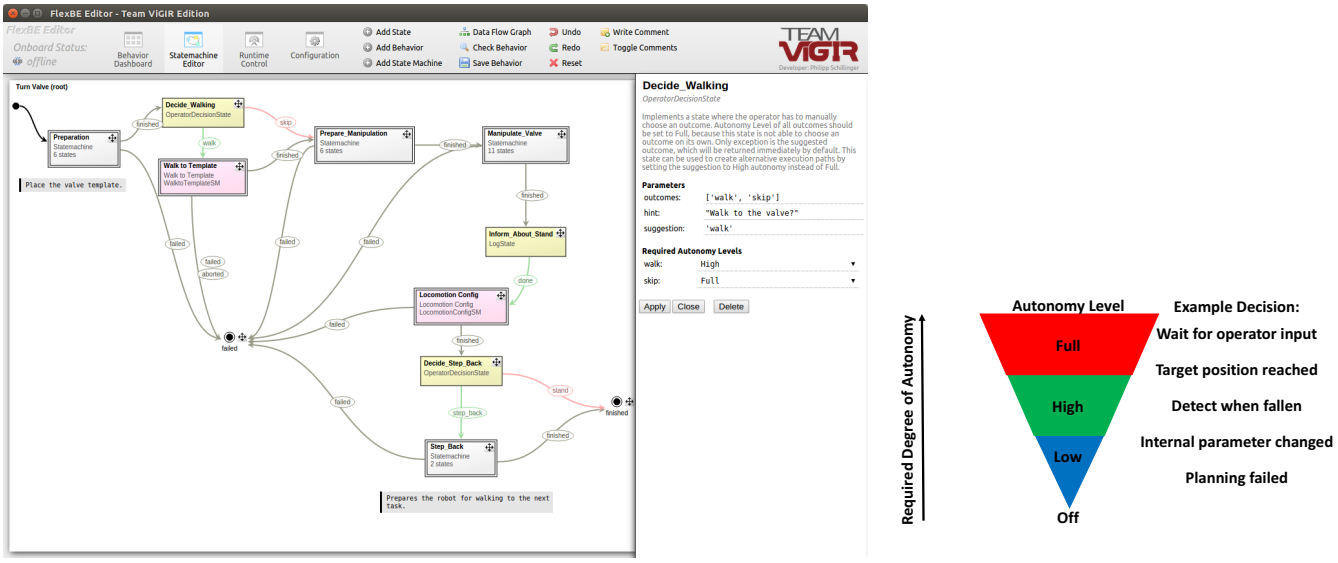
A major extension to SMACH is that FlexBE enables runtime modifications to existing behaviors[5]. FlexBE implements a *lockable state* that maintains consistency during runtime modifications. This capability has been demonstrated with manual modifications via the FlexBE UI[5] and with automatic behavior synthesis[35].

A key design motivation for FlexBE is to move beyond simple state machine execution into *collaborative autonomy*[3]. This concept moves beyond basic sliding or adjustable autonomy[36] for control and decision making to include *collaborative perception* where the operator can assist the robot in perception. For example, the robot can request that the operator add virtual information such as placement of a valve template or goal location[37], [38]. FlexBE is designed to support this concept of collaborative autonomy with bi-directional data flow. FlexBE also allows the operator to monitor state transitions, and can require the operator to confirm certain transitions. In this way, the state machines themselves act as a script to guide the operator through challenging tasks and reduce cognitive burden[3].

The FlexBE system provides significant flexibility for many tasks, but the current ROS Navigation stack as described in Section II-B precludes this level of operator-robot interaction for mobile robot navigation as many decision states are embedded inside move_base. This motivates the primary contribution of this paper, which we call *Flexible Navigation*.

IV. Flexible Navigation

This paper describes our open source software that extends the ROS Navigation system to enable collaborative autonomy with moving mobile robotic platforms. Figure 5 shows an



(a) FlexBE turn valve behavior (best viewed online with magnification)

(b) Adjustable autonomy levels

Fig. 4. FlexBE usage by Team ViGIR in DARPA Robotics Challenge

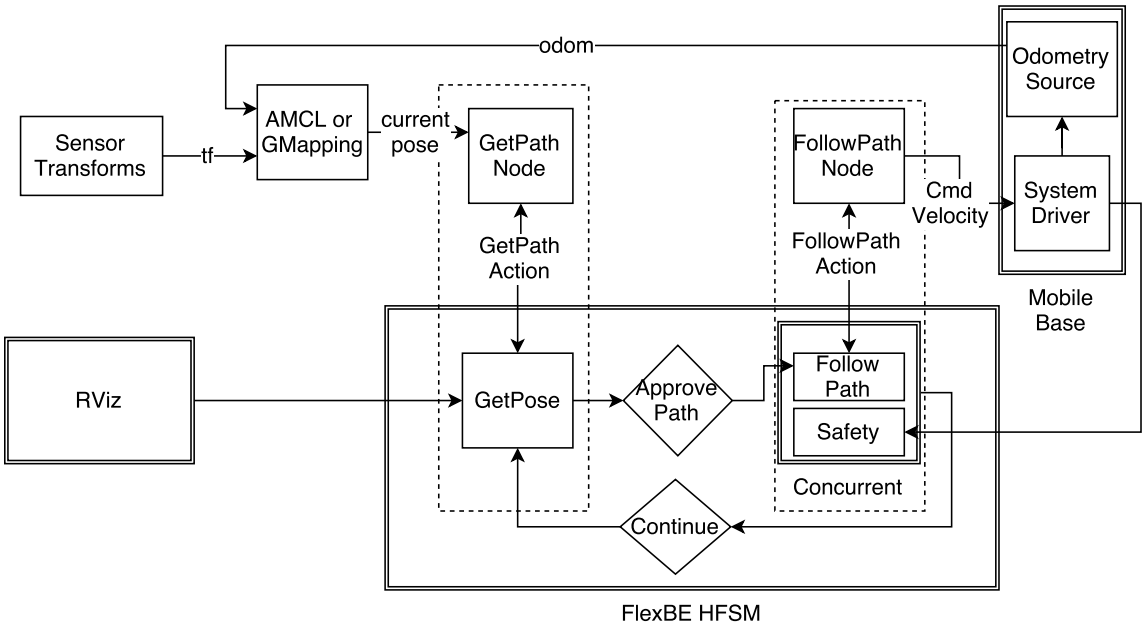


Fig. 5. One simple configuration using the Flexible Navigation architecture.

example block diagram that can be contrasted with Figure 2. We begin with a description of the system design goals and architecture, and then describe the open source code and demonstrations.

A. Design

A key design goal for Flexible Navigation was to preserve compatibility with ROS Navigation in order to leverage ongoing work in planner improvement, and ensure future extensibility. Our approach decomposed the integrated move_base into multiple nodes that implement the ROS Navigation base_global_planner[22] and base_local_planner[23] plugins. By decomposing global and local planners, we provide the designer with flexibility to develop multi-layered planning scenarios, and allow a supervisor to choose among multiple plans, or provide post-processing of the plans.

Another key design goal was to move toward ROS ActionLib[24] interfaces for the planning nodes as a natural interface to the HFSM, and to provide simple message passing with preemption that allows planning or execution to be canceled or

[22] http://wiki.ros.org/global_planner?distro=kinetic
[23] http://wiki.ros.org/base_local_planner?distro=kinetic
[24] http://wiki.ros.org/actionlib

preempted. The Flexible Navigation system provides FlexBE state implementations that interface with the all of the planning nodes via an ActionLib client interface. Additional state implementations are described below.

In keeping with the HFSM-based design, Flexible Navigation implements recovery behaviors as hierarchical state machines. This provides flexibility to the designer to reuse existing behaviors, create new ones, or adapt on the fly using FlexBE. Ultimately, the Flexible Navigation system provides design freedom by using existing software from ROS Navigation, while supporting the system designer's ability to specify behaviors as needed.

FlexBE has a special concurrent state implementation that supports parallel execution of states; this is useful for safety and health monitoring while the system is following a path. Any of the parallel states can trigger completion or failure, and preempt the current follow action. Figure 6 shows the complete HFSM used in this paper's demonstrations.

The core open source Flexible Navigation[25] software system defines a collection of ROS packages.

1) flex_nav_common: This package defines a collection of actionlib actions used by the state implementations.

GetPath action, which uses a goal PoseStamped[26] data structure, returns the resulting Path[27] and defined status code as the result. The action returns the current system Pose[28] and goal Pose as feedback during planning.

FollowPath action uses a goal Path, and reports the current system Pose and Twist[29] velocity as feedback during execution, before returning the resulting final Pose and status code.

FollowTopic action uses a topic String[30] that the node should listen for to receive an updated Path, and returns the final Pose and status code as result. The action reports the current system Pose and current Path being followed as feedback during execution.

ClearCostmap action causes the action server to clear the current cost map; this is used as part of a recovery behavior.

2) flex_nav_planners: This package defines ROS nodes that use ROS Navigation base_global_planner and 2D costmapper plugins, and plans a path from current system Pose to a defined goal. The nodes differ on the action server that is implemented, and how the goal is defined.

GetPath (get_path_node) implements a GetPath action server. The system plans a path to the defined action goal, and returns the resulting path; the node also publishes the planned path as a ROS topic. This node is generally used to return a path for inspection and approval prior to execution.

FollowPath (follow_path_node) implements a FollowPath action server. The system looks ahead along the goal path a given distance, and plans a new local path to the look ahead point. As the system moves, the look ahead point is updated and a new plan is generated. This effectively functions as the standard ROS Navigation base_local_planner, but allows a richer set of planners. This node does not publish the command velocity; a local path following controller is needed for that.

FollowTopic (follow_topic_node) implements a FollowTopic action server, but otherwise functions the same as the FollowPath node.

3) flex_nav_controllers: This package defines a collection of ROS nodes that function as local path following controllers. The follow_path_node and follow_topic_nodes provide the respective FollowPath and FollowTopic action servers, and implement the ROS base_local_planner and 2D costmapper plugins to mimic the move_base local planner. In a change from the standard move_base, the controller nodes publish the command velocity as a TwistStamped[31] message to the robot, which improves data logging and debugging; it is trivial to define a relay that strips the timestamp from the message to connect to standard drivers.

The package also provides two scripts flex_nav_controller_pure_pursuit_{path, topic} that implement the pure pursuit algorithm as their corresponding action servers[39].

4) flex_nav_flexbe_states: This package provides a number of state implementations that provide action clients for GetPath, FollowPath, FollowTopic, and ClearCostmap actions. These interface with the nodes from flex_nav_planners and flex_nav_controllers packages. A GetPose state implementation gets a PoseStamped from a topic (e.g. via RViz 2D navigation goal) and defines the corresponding GetPath action. There is a simple python implementation of the pure pursuit algorithm as a state that gets its path from FlexBE user data defined by the input transition. Simple states for moving a given distance and rotating through a given angle are also provided; these can be used with recovery behaviors.

B. Open Source

In addition to the base Flexible Navigation packages, a collection of additional repositories are open source on GitHub[32]. These include a few customized forks from other repos, and some original packages that support the integrated demonstration. The code is developed and tested using the ROS Kinetic release, but should be compatible with earlier versions with minimal changes.

The two primary repositories that contain complete information about setting up and running the demonstrations are chris_create_flexible_navigation[33] and chris_turtlebot_flexible_navigation[34]. These repositories have README files that display installation and startup instructions for the demonstrations using either an iRobot Create-based platform or a Kobuki-based Turtlebot platform. The repositories contain packages with specific state

[25]https://github.com/CNURobotics/flexible_navigation
[26]http://docs.ros.org/api/geometry_msgs/html/msg/PoseStamped.html
[27]http://docs.ros.org/api/nav_msgs/html/msg/Path.html
[28]http://docs.ros.org/api/geometry_msgs/html/msg/Pose.html
[29]http://docs.ros.org/api/geometry_msgs/html/msg/Twist.html
[30]http://docs.ros.org/api/std_msgs/html/msg/String.html
[31]http://docs.ros.org/api/geometry_msgs/html/msg/TwistStamped.html
[32]https://github.com/CNURobotics
[33]https://github.com/CNURobotics/chris_create_flexible_navigation
[34]https://github.com/CNURobotics/chris_turtlebot_flexible_navigation

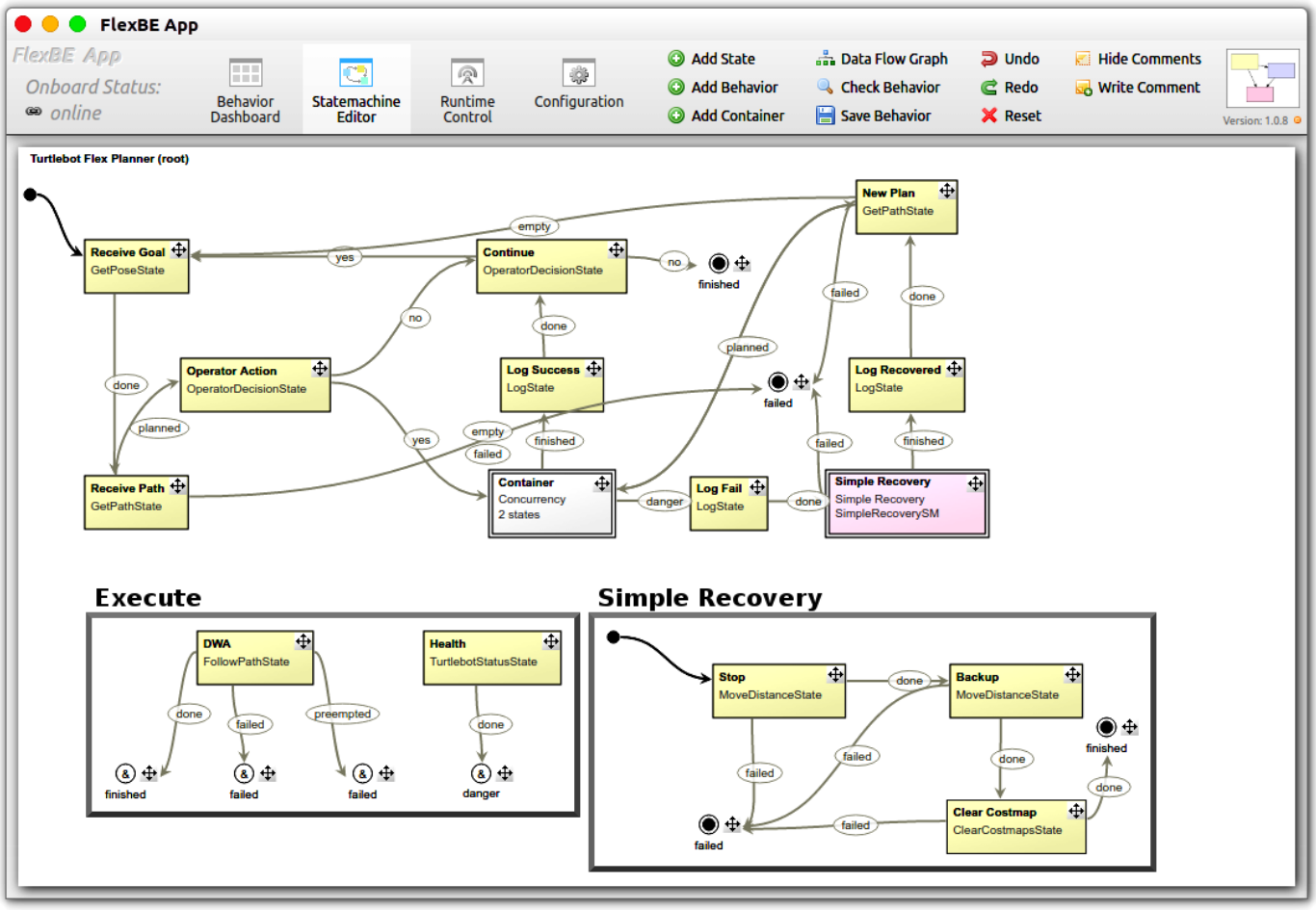


Fig. 6. Flexible Navigation system that mimics move_base while supporting collaborative autonomy. The concurrent Navigate state provides local planning and robot control while monitoring robot safety.

implementations that interface with robot specific messages, and provide launch files for the consistent demonstrations.

Both of the robot specific setups use the chris_install[35] repository, which provides scripts based on ROS Workspace tools[36] that manage installation of multiple repositories. Specific directions for both the Create and Turtlebot setups are provided on their respective flexible_navigation repositories.

Both of the Create and Turtlebot setups are customized versions based on the ROS Turtlebot[37] software. In addition to support of the TwistStamped command velocities, the systems are modified to support namespaced invocations that support multi-robot scenarios on a single ROS network. For specific version information, consult the associated rosinstall files that define the relevant repositories and consistent branches for each package.

[35]https://github.com/CNURobotics/chris_install
[36]http://wiki.ros.org/wstool
[37]https://github.com/turtlebot

V. Experimental Validation

This paper demonstrates the Flexible Navigation system in both simulation and hardware, using the same code and directions that are open sourced. Figure 6 shows the specific state machine used in both demonstrations. Both the simulation and hardware demonstrations use a Kobuki-based Turtlebot with Microsoft Kinect sensor and a Hokuyo URG-04lx LIDAR sensor; only the LIDAR scan was used for AMCL-based localization using a known map of the environment. The same navigation and control software is used in both demonstrations.

A. Simulation

The simulations were performed using Gazebo 7 released as part of the ROS Kinetic distribution. Figure 7 shows the model of an obstacle range in our lab; Figure 3 shows the corresponding occupancy (cost) map after one lap around the simulated world. In this case, we repeated the input of three intermediate goals as the system was operating in order to complete the loop [38]. During the demonstration, one bumper

[38]Our videos can be found at https://www.youtube.com/playlist?list=PLg9fIiOWSqkF27QpB3nUTO9U2XXkX3jBV.

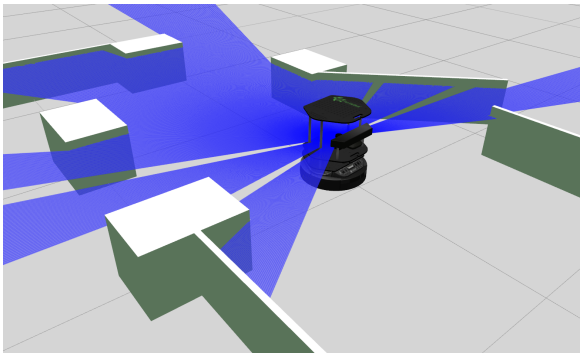


Fig. 7. Simulation of turtlebot in obstacle course. The simulation shows the simulated Hokuyo LIDAR beams.

activation was simulated to initiate the recovery behavior shown in Figure 6, which causes the robot to stop, back up 0.025 meters, and clear the cost map. The robot then replans to the original goal.

B. Hardware

The hardware demonstration used the Flexible Navigation system with a customized Turtlebot setup[39]. In this demonstration we tested the recovery behaviors after contact with the front bumper. Figure 1 shows the robot navigating the obstacle course during operation.

VI. Conclusion

This paper introduces the Flexible Navigation system that extends the existing ROS Navigation system to work with FlexBE to provide hierarchical finite state machine based collaborative autonomy for navigating mobile robotic platforms. This approach provides improved flexibility and user control, while preserving future extensibility through compatibility with ROS Navigation's move_base compatible plugins. The project is available open source, and can be easily demonstrated on a desktop computer with an integrated simulation using either iRobot Create or Turtlebot platforms; comprehensive installation and testing instructions are provided online for two integrated demonstrations[40]. The approach can be applied to any system that uses ROS Navigation's move_base.

Future work will integrate a route planner into the Flexible Navigation system, improve the motion primitives used by the planners, and provide more built in state implementations for recovery behaviors. On going research will build upon the synthesis capabilities to provide for automatic generation of recovery behaviors when existing behaviors fail[35].

References

[1] A. Stentz *et al.*, "CHIMP, the CMU Highly Intelligent Mobile Platform," *Journal of Field Robotics*, vol. 32, no. 2, pp. 209–228, 2015, ISSN: 1556-4967. DOI: 10.1002/rob.21569.

[2] M. DeDonato *et al.*, "Human-in-the-loop Control of a Humanoid Robot for Disaster Response: A Report from the DARPA Robotics Challenge Trials," *Journal of Field Robotics*, vol. 32, no. 2, pp. 275–292, 2015, ISSN: 1556-4967. DOI: 10.1002/rob.21567.

[3] A. Romay *et al.*, "Collaborative autonomy between high-level behaviors and human operators for remote manipulation tasks using different humanoid robots," *Journal of Field Robotics*, n/a–n/a, 2016, ISSN: 1556-4967. DOI: 10.1002/rob.21671.

[4] E. Marder-Eppstein *et al.*, "The Office Marathon: Robust navigation in an indoor office environment," in *2010 IEEE International Conference on Robotics and Automation*, May 2010, pp. 300–307. DOI: 10.1109/ROBOT.2010.5509725.

[5] P. Schillinger, S. Kohlbrecher, and O. von Stryk, "Human-robot collaborative high-level control with application to rescue robotics," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, May 2016, pp. 2796–2802. DOI: 10.1109/ICRA.2016.7487442.

[6] R. C. Arkin and T. Balch, "AuRA: Principles and Practice in Review," *Journal of Experimental and Theoretical Artificial Intelligence*, vol. 9, pp. 175–189, 1997.

[7] S. Thrun *et al.*, "Stanley: The robot that won the DARPA Grand Challenge," *Journal of Field Robotics*, vol. 23, no. 9, pp. 661–692, 2006, ISSN: 1556-4967. DOI: 10.1002/rob.20147.

[8] C. Urmson *et al.*, "Autonomous driving in urban environments: Boss and the Urban Challenge," *Journal of Field Robotics*, vol. 25, no. 8, pp. 425–466, 2008, ISSN: 1556-4967. DOI: 10.1002/rob.20255.

[9] M. Montemerlo *et al.*, "Junior: The Stanford entry in the Urban Challenge," *Journal of Field Robotics*, vol. 25, no. 9, pp. 569–597, 2008, ISSN: 1556-4967. DOI: 10.1002/rob.20258.

[10] A. Bacha *et al.*, "Odin: Team VictorTango's entry in the DARPA Urban Challenge," *Journal of Field Robotics*, vol. 25, no. 8, pp. 467–492, 2008, ISSN: 1556-4967. DOI: 10.1002/rob.20248.

[11] E. Krotkov *et al.*, "The DARPA Robotics Challenge Finals: Results and Perspectives," *Journal of Field Robotics*, n/a–n/a, 2016, ISSN: 1556-4967. DOI: 10.1002/rob.21683.

[12] M. Johnson *et al.*, "Team IHMC's Lessons Learned from the DARPA Robotics Challenge: Finding Data in the Rubble," *Journal of Field Robotics*, n/a–n/a, 2016, ISSN: 1556-4967. DOI: 10.1002/rob.21674.

[13] M. Fallon *et al.*, "An Architecture for Online Affordance-based Perception and Whole-body Planning," *Journal of Field Robotics*, vol. 32, no. 2, pp. 229–254, 2015, ISSN: 1556-4967. DOI: 10.1002/rob.21546.

[14] B. Gerkey. (). ROS & Gazebo at the DRC Finals, [Online]. Available: http://www.osrfoundation.org/ros-gazebo-at-the-drc-finals/ (visited on 01/06/2017).

[39] https://github.com/CNURobotics/chris_ros_turtlebot

[40] chris_create_flexible_navigation and chris_turtlebot_flexible_navigation

[15] M. Quigley *et al.*, "ROS: an open-source Robot Operating System," in *ICRA Workshop on Open Source Software*, 2009.

[16] M. Quigley, B. Gerkey, and W. D. Smart, *PROGRAMMING ROBOTS WITH ROS: A PRACTICAL INTRODUCTION TO THE ROBOT OPERATING SYSTEM*. Sebastopol, CA: O'Reilly Media, 2015, ISBN: 9781449323882.

[17] J. Boren and S. Cousins, "Exponential Growth of ROS [ROS Topics]," *IEEE Robotics Automation Magazine*, vol. 18, no. 1, pp. 19–20, Mar. 2011, ISSN: 1070-9932. DOI: 10.1109/MRA.2010.940147.

[18] C. Aguero *et al.*, "Inside the Virtual Robotics Challenge: Simulating Real-Time Robotic Disaster Response," *Automation Science and Engineering, IEEE Transactions on*, vol. 12, no. 2, pp. 494–506, Apr. 2015, ISSN: 1545-5955. DOI: 10.1109/TASE.2014.2368997.

[19] J. M. O'Kane, *A GENTLE INTRODUCTION TO ROS*. Independently published, Oct. 2013, Available at http://www.cse.sc.edu/~jokane/agitr/, ISBN: 978-1492143239.

[20] E. Fernandez *et al.*, *LEARNING ROS FOR ROBOTICS PROGRAMMING - SECOND EDITION*, 2nd. Packt Publishing, 2015, ISBN: 1783987588, 9781783987580.

[21] R. B. Rusu and S. Cousins, "3D is here: Point Cloud Library (PCL)," in *2011 IEEE International Conference on Robotics and Automation*, May 2011, pp. 1–4. DOI: 10.1109/ICRA.2011.5980567.

[22] I. A. Sucan, M. Moll, and L. E. Kavraki, "The Open Motion Planning Library," *IEEE Robotics Automation Magazine*, vol. 19, no. 4, pp. 72–82, Dec. 2012, ISSN: 1070-9932. DOI: 10.1109/MRA.2012.2205651.

[23] M. Likhachev. (). Search Based Planning Library, [Online]. Available: http://sbpl.net/Software (visited on 01/05/2017).

[24] S. Chitta, I. Sucan, and S. Cousins, "MoveIt! [ROS Topics]," *IEEE Robotics Automation Magazine*, vol. 19, no. 1, pp. 18–19, Mar. 2012, ISSN: 1070-9932. DOI: 10.1109/MRA.2011.2181749.

[25] R. Reid *et al.*, "Cooperative multi-robot navigation, exploration, mapping and object detection with ROS," in *2013 IEEE Intelligent Vehicles Symposium (IV)*, Jun. 2013, pp. 1083–1088. DOI: 10.1109/IVS.2013.6629610.

[26] D. V. Lu, D. Hershberger, and W. D. Smart, "Layered costmaps for context-sensitive navigation," in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Sep. 2014, pp. 709–715. DOI: 10.1109/IROS.2014.6942636.

[27] Q. Xu *et al.*, "Design and implementation of an ROS based autonomous navigation system," in *2015 IEEE International Conference on Mechatronics and Automation (ICMA)*, Aug. 2015, pp. 2220–2225. DOI: 10.1109/ICMA.2015.7237831.

[28] T. Krajnk *et al.*, "Persistent localization and life-long mapping in changing environments using the Frequency Map Enhancement," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Oct. 2016, pp. 4558–4563. DOI: 10.1109/IROS.2016.7759671.

[29] D. Fox, "KLD-Sampling: Adaptive Particle Filters," in *Advances in Neural Information Processing Systems 14*, MIT Press, 2001.

[30] G. Grisetti, C. Stachniss, and W. Burgard, "Improved techniques for grid mapping with Rao-Blackwellized particle filters," *IEEE Transactions on Robotics*, vol. 23, no. 1, pp. 34–46, Feb. 2007, ISSN: 1552-3098. DOI: 10.1109/TRO.2006.889486.

[31] M. Pivtoraiko, R. A. Knepper, and A. Kelly, "Differentially constrained mobile robot motion planning in state lattices," *Journal of Field Robotics*, vol. 26, no. 3, pp. 308–333, Mar. 2009.

[32] J. Bohren and S. Cousins, "The SMACH High-Level Executive [ROS News]," *IEEE Robotics Automation Magazine*, vol. 17, no. 4, pp. 18–20, Dec. 2010, ISSN: 1070-9932. DOI: 10.1109/MRA.2010.938836.

[33] S. Kohlbrecher *et al.*, "Human-robot teaming for rescue missions: Team ViGIR's approach to the 2013 DARPA Robotics Challenge Trials," *Journal of Field Robotics*, vol. 32, no. 3, pp. 352–377, 2015, ISSN: 1556-4967. DOI: 10.1002/rob.21558.

[34] S. Kohlbrecher *et al.*, "A comprehensive software framework for complex locomotion and manipulation tasks applicable to different types of humanoid robots," *Frontiers in Robotics and AI*, vol. 3, no. 31, 2016, ISSN: 2296-9144. DOI: 10.3389/frobt.2016.00031.

[35] S. Maniatopoulos *et al.*, "Reactive high-level behavior synthesis for an Atlas humanoid robot," in *Proceedings of the 2016 IEEE International Conference on Robotics and Automation (ICRA '16)*, May 2016.

[36] J. W. Crandall and M. A. Goodrich, "Experiments in adjustable autonomy," in *2001 IEEE International Conference on Systems, Man and Cybernetics. e-Systems and e-Man for Cybernetics in Cyberspace (Cat.No.01CH37236)*, vol. 3, 2001, 1624–1629 vol.3. DOI: 10.1109/ICSMC.2001.973517.

[37] A. Romay *et al.*, "Achieving versatile manipulation tasks with unknown objects by supervised humanoid robots based on object templates," in *Humanoid Robots (Humanoids), 2015 IEEE-RAS 15th International Conference on*, Nov. 2015, pp. 249–255. DOI: 10.1109/HUMANOIDS.2015.7363543.

[38] A. Stumpf *et al.*, "Open source integrated 3D footstep planning framework for humanoid robots," in *2016 IEEE-RAS 16th International Conference on Humanoid Robots (Humanoids)*, Nov. 2016, pp. 938–945. DOI: 10.1109/HUMANOIDS.2016.7803385.

[39] R. C. Coulter, "Implementation of the pure pursuit path tracking algorithm," Robotics Institute, Pittsburgh, PA, Tech. Rep. CMU-RI-TR-92-01, Jan. 1992.