Towards autonomous robotic butlers: Lessons learned with the PR2

Conference Paper *in* Proceedings - IEEE International Conference on Robotics and Automation · May 2011

DOI: 10.1109/ICRA.2011.5980058 · Source: DBLP

9 authors, including:

Jonathan Bohren
Johns Hopkins University
6 PUBLICATIONS **836** CITATIONS
SEE PROFILE

Radu Bogdan Rusu
Open Perception
68 PUBLICATIONS **18,485** CITATIONS
SEE PROFILE

Caroline Pantofaru
Google Inc.
39 PUBLICATIONS **3,611** CITATIONS
SEE PROFILE

Stefan Holzer
Technical University of Munich
15 PUBLICATIONS **3,133** CITATIONS
SEE PROFILE

Towards Autonomous Robotic Butlers: Lessons Learned with the PR2

Jonathan Bohren, Radu Bogdan Rusu, E. Gil Jones, Eitan Marder-Eppstein, Caroline Pantofaru,
Melonee Wise, Lorenz Mösenlechner, Wim Meeussen, Stefan Holzer
Willow Garage
68 Willow Rd., Menlo Park, CA 94025, USA
{jbohren,rusu,gjones,eitan,pantofaru,mwise,lorenz,meeussen,holzers}@willowgarage.com

***Abstract*— As autonomous personal robots come of age, we expect certain applications to be executed with a high degree of repeatability and robustness. In order to explore these applications and their challenges, we need tools and strategies that allow us to develop them rapidly. Serving drinks (i.e., locating, fetching, and delivering), is one such application with well-defined environments for operation, requirements for human interfacing, and metrics for successful completion. In this paper we present our experiences and results while building an autonomous robotic assistant using the PR2[1] platform and ROS[2]. The system integrates several new components that are built on top of the PR2's current capabilities. Perception components include dynamic obstacle identification, mechanisms for identifying the refrigerator, types of drinks, and human faces. Planning components include navigation, arm motion planning with goal and path constraints, and grasping modules. One of the main contributions of this paper is a new task-level executive system, *SMACH*, based on hierarchical concurrent state machines, which controls the overall behavior of the system. We provide in-depth discussions on the solutions that we found in accomplishing our goal, and the implementation strategies that let us achieve them.**

I. Introduction

Building a broadly capable, autonomous robotic platform is a near-universal goal among roboticists, but the current generation of technical systems fall short of this goal. A potential step along the way towards broad capability is to create a large set of specific, useful robotic behaviors that can accomplish particular goals given a well-defined, constrained application area. Such a set could be considered an "application store" for personal robots; these apps may lack capability across application boundaries but can still be usefully employed in particular circumstances. This view of encapsulating particular functionality is gaining acceptance across the research community [1] [2] [3] and is an important component for the near and long term visions of endowing personal robots with abilities that will better assist people.

Even in the relatively well-constrained bounds of a specific "application", endowing a personal robot with autonomous capability will require integrating many complex subsystems; most robots will need some facility in perception, motion planning, reasoning, navigation, and grasping. Though these individual components may have been extensively validated in isolation, seamlessly integrating the components into a robust heterogeneous system is still an active area of research. Specific challenges integrators face include coping with multiple points of failure from complicated subsystems, computational constraints that may only be encountered when running large sets of components, and reduced performance from subsystems during interoperation. Furthermore, subsystem designers may not have anticipated all the criteria required for successful application performance, and may not have tested extensively for particular circumstances that will be encountered during application operation.

Fig. 1. The PR2 robot fetching a drink from the refrigerator.

A final challenge for the implementors of applications is not just in creating robust capabilities but enabling extension and reuse of those capabilities in future applications. For the "app store" paradigm to succeed in moving towards broad capability, easy extendibility and interoperation of applications is essential. This requirement necessitates clean interfaces that allow for oversight by an executive. Furthermore, it should also be straightforward to contribute training data or new components to the system without requiring substantial redesign. For example, one may wish to extend an application to a new environment, enable the robot to cope with a different set of objects, or swap out an object manipulation subsystem with a more capable system; making such changes would ideally require little additional work or understanding the complete operation of the application.

In this work we focus on one particular application for a personal robot: fetching and serving drinks. In addition to improving on the state-of-the-art in drink fetching by a

[1]PR2 (Personal Robot 2) is a robotic platform developed by Willow Garage – http://www.willowgarage.com

[2]ROS (Robot Operating System) is an open source, software initiative supported by an international community of robotics researchers – http://www.ros.org

personal robot, we introduce a novel task-level executive, *SMACH*, based on hierarchical concurrent state machines. We argue that *SMACH*, when used with ROS, allows applications such as this one to be developed more rapidly than with imperative scripting approaches or model-based task planners. Additionally, *SMACH* is designed to work with task-planning systems, as a procedure-definition architecture.

In this paper we focus on both the subsystems that enable the drink fetching application as well as the use of SMACH as a system for rapid and reusable application development, integration, and execution. Section III details the application architecture, navigation capabilities, and the drink ordering graphical user interface. In Section IV we discuss the perceptual subsystems, including mechanisms for identifying the refrigerator, types of drinks, grasp positions, human faces, and creating dynamic obstacle maps. Section V discusses the use of arm motion planning with goal and path constraints in the presence of dynamic obstacles; these capabilities are used for refrigerator door opening and closing as well as drink fetching, stowing, and safe delivery and handoff. In Section VI we describe SMACH and detail its use in the drink fetching application. We conclude with results and discussion in Section VII.

II. Related Work

Several efforts have been already made regarding the creation of complete autonomous robotic systems that can serve drinks. The work in [4] describes Justin, a robotic platform with two 7-DOF arms, 5-DOF torso, and 4-finger 12-DOF hands. Justin can detect glasses and bottles present on tables in its field of view, has the ability to open certain types of bottles by unscrewing the cap, and can pour drinks into a glass. The authors focus on dexterous manipulation with the 4-fingered hand plus object categorization and identify individual elements rather than detecting different types of objects, such as bottle types, etc. In [5], the Care-O-Bot robotic system with one arm and a tray is presented as a demonstrator for serving drinks. The robot can detect certain drinks and place them on a tray, and the user can interact with the system via a touch screen. The authors analyze the overall performance of the system and report an overall success rate of 40%. This is a strong indicator which agrees with our assessment in Section I regarding complex systems. Another similar one-arm robotic system based on a Segway platform (HERB) is presented in [6]. The authors tackle the manipulation of doors, handles, cabinets in the presence of clutter. However, localization is solved through the use of checkerboard patterns installed in the world, and manipulation is mostly performed on tabletops. A semi-autonomous mobile robot that can take drink orders, without manipulation capabilities, is presented in [7]. The robot takes drink orders from an user, moves to a bartender, and the bartender puts a drink on the robot. Then, the robot drives back to the user, which has to take the drink from the robot. This is an interesting approach, but it falls into a different category of systems which do not posses full autonomy. The work in [8] describes ARMAR-III, a general purpose robot with two 7 DOF arms and a 3 DOF torso. The authors describe the hardware and the perception system, but do not provide data regarding the experimental setup, or whether they are attempting to use the robot for object delivery. Another system presented in [9] can extract a drink from a fridge with supervised autonomy. The supervision comes in mostly in the context of verifying the output of various perception subsystems. The remote supervisor sits at a terminal and is prompted at all decision points to either continue, retry, or abort. This implementation basically avoids the complex failure recovery logic that makes developing these applications quickly so challenging. Our strategy for developing applications of this sort attempts to address all the above mentioned challenges.

III. Architecture

Our experimental system architecture is building upon our previous work [10], [11], where we tackled the problems of navigation, door identification and opening, and plugging in. The hardware and software platforms used are the PR2 robot (as shown in Figure 1) and ROS – see [12] for a brief hardware and software description.

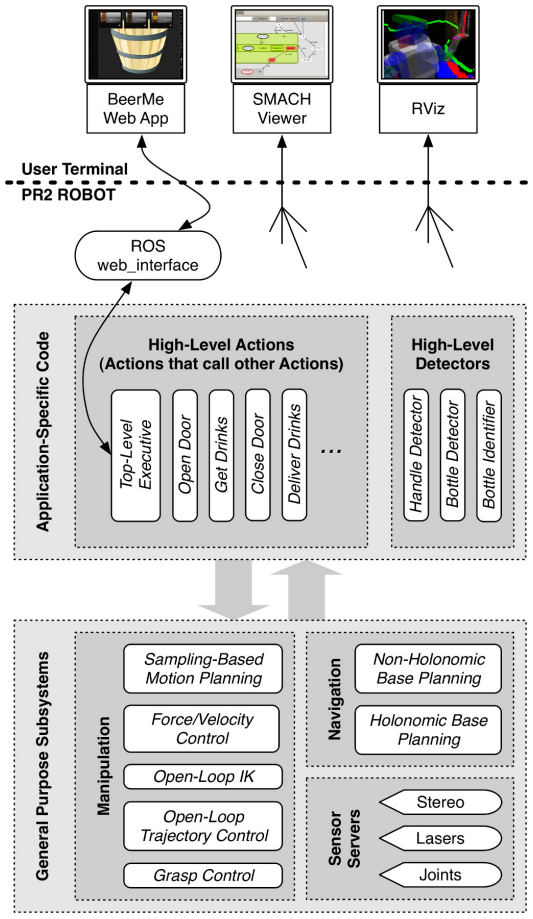

Fig. 2. The overall architecture of our system using ROS, including the three major modules together with their individual components: i) general purpose subsystems; ii) application-specific code; and iii) user interfaces.

The modular structure of our framework allows seamlessly swapping components that share the same interface. We

identify three major types of component classes, namely: i) *general purpose subsystems* – contain code that is not application specific and can be reused by other applications; ii) *application-specific code* – represents the collection of sensor data processing and higher level actions that are specific for a particular application; and finally iii) *user interfaces*. Figure 2 presents an overview on the type of components that are particular to each of the above mentioned classes.

The first category includes basic sensor and actuator data producers and consumers, as well as navigation and motion planning algorithms. Examples of the latter are inverse kinematics modules, trajectory controllers, grasping methods, etc. The second category is specific to our application and includes perception detectors for drinks, human face detection, and handle and bottle grasp point extraction methods. Finally in the third category, we provide user control through a web interface, as well as other visualization tools.

The robot navigation stack [10] uses sensor data from laser range finders mounted on the PR2 to build a 3D voxel grid representing obstacle data in the world as well as tracking unknown space. This voxel grid is then projected down to 2D where it is used for planning. The stack effectively avoids obstacles typically found in an office environment such as tables, chairs and people, allows the robot to move safely in its environment, and is also responsible for localization. This allows the robot to navigate to its set of delivery locations, as well as the refrigerator location, by using poses stored in a global frame.

IV. Perception

The perceptual capabilities of our system include several components, naming: i) a Dynamic Obstacle Map method which is used by the motion planning algorithms to compute safe arm trajectories to the goal; ii) refrigerator door and handle identification, used to estimate a grasping strategy that can open the door; iii) a drinks and grasping pose identification module; and finally iv) human face detection during the delivery phase. In the following we will address each of these components individually.

A. Dynamic Obstacle Maps

In order to move the arms safely in the environment, the robot must be able to anticipate not only self-collisions between the robot's limbs and its body, but also the environment around the robot. For the latter purpose, the robot must use its sensors to find environment obstacles so that motion planning can be used to achieve goals while avoiding collisions.

For this application we used two primary sources of environment collision data: the tilting laser data and stereo camera data. The tilting laser provides data in a wide field of view - up to 270° - but provides 3D data slowly as the laser must be actuated in pitch. A further limitation is that the laser is fixed to the neck of the robot and thus cannot be targeted at particular locations of interest by pointing the head. We supplement the tilting laser data with stereo point clouds from the narrow stereo pair on the PR2's head. The stereo point clouds are dense, especially when paired with a texture projector, and are available at up to 30Hz; however, the data has a very limited field of view, and so care must be taken to point the head to the most relevant places. The general strategy for composing collision maps is to use the tilting laser to perceive macro environment structures like tables, chairs, people or refrigerator doors, and then to point the head at particular areas of interest, such as targets of manipulation.

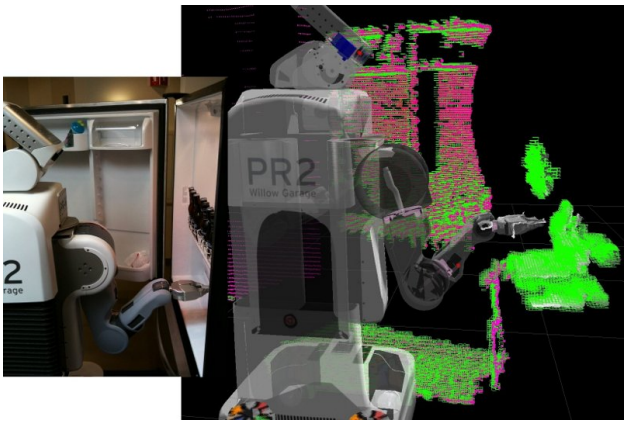

Fig. 3. A visualization screenshot of the dynamic obstacle map produced in the inset robot configuration. Both the laser (purple) and stereo points clouds (grey) are shown, with the voxels of the collision map shown in green. Note that the stereo points on the robot's hand are removed by self-filtering and thus not included in the dynamic obstacle map.

The sensor processing pipeline first filters the data streams for shadow points as well as points that belong to the robot's body - these points should not be avoided as environmental obstacles. The data from the two streams is then passed into the dynamic obstacle map generation code. The dynamic obstacle map is comprised of a set of voxel primitives implemented as flat 3D arrays [12]. For this application we used oriented bounding boxes to represent occupied voxels, with a voxel resolution of 1 cm. The two sensor streams are fused using a union of occupied cells. The workspace of potential occupancy is a fixed size around the robot, large enough to contain any points that could be contacted by the robot arms given a fixed-base location. Any area not currently perceived as occupied is assumed to be obstacle-free. An example of the dynamic obstacle map produced when the robot is in position to grab a beverage is shown in Figure 3.

B. Fridge Door and Handle Identification

Due to the large space of all possible refrigerator configurations, we are constraining the fridge identification problem to the following: i) the fridge door has a vertical handle which can be pulled sideways; ii) an approximate position of the fridge location is given by the navigation system.

Our fridge door and handle identification method follows a pattern similar to the one that we previously proposed in [13]. In detail, we make use of the dense stereo point cloud $\mathcal{P}$ obtained from the PR2's narrow stereo camera pair and perform the following computational steps:

- create a downsampled representation $\mathcal{P}_d$ of $\mathcal{P}$ using a voxel grid filter;
- segment the closest dominant near-vertical plane in $\mathcal{P}_d$ using a sample consensus oriented plane fitting method;
- compute the convex hull of the inliers of the plane, and remove all points which are outside of the polygonal prism created from the plane having a height equal with the distance to the plane;
- extract all the points $\boldsymbol{p}_i \in \mathcal{P}_d$ which lie between the camera and the dominant plane, i.e.,

$$\|V_p - \boldsymbol{p}_i\|_2 < \|V_p - \boldsymbol{p}_i^p\|_2 \quad (1)$$

  where V_p represents the camera viewpoint, and $\boldsymbol{p}_i^p$ represents the projection of $\boldsymbol{p}_i$ on the plane;
- perform clustering on the points $\boldsymbol{p}_i$ using a Euclidean clustering algorithm;
- select the biggest cluster, estimate the cluster centroid $\overline{\boldsymbol{p}_i}$ of $\boldsymbol{p}_i$, and the primary axis of the cluster.

The handle grasp point is then the centroid of the estimated cluster, oriented along the cross product of the fridge door normal and the handle primary axis. Figure 4 presents an example of the detection results. The above mentioned filters and algorithms have been implemented in PCL [14].

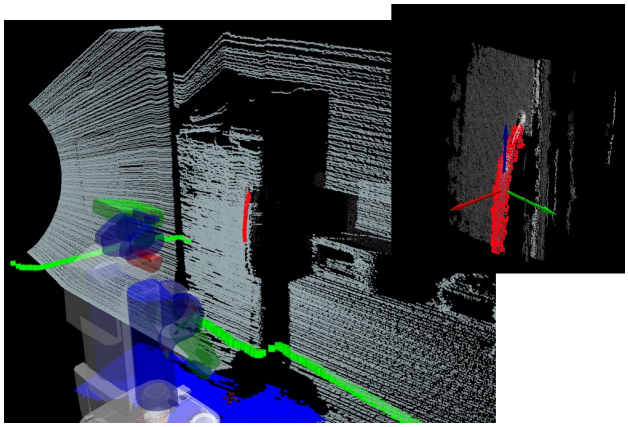

Fig. 4. The output of the fridge handle detector from the narrow stereo, showing the handle points (red) and the handle orientation (top right). The left part of the figure presents the point cloud obtained using the tilting laser sensor, with a transparent model of the PR2 robot in the foreground.

C. Drinks Identification and Grasping Pose Estimation

Most bottled drinks can be identified by a label with a somewhat unique signature which usually occupies at least one third of the space on a bottle. Given that the label is the only information that can help in recognizing the particular type of drink, our identification method relies on visual texture cues, i.e., 2D feature matching. This satisfies our application's perception constraints. Specifically, adding a new type of drink to be identified should be as easy as possible, without requiring a 3D model capture or a multitude of template views. Moreover, we cannot guarantee that the orientation of a bottle is always facing with the label towards the camera, and as the orientation is not required for grasping, our only constraint is that a large enough patch of the label has to be visible for feature matching.

A 2D descriptor that was proven to work reliably for object recognition in the computer vision literature is SIFT [15]. Although SIFT is computationally expensive, this limitation does not constitute a problem for our application, since the bottles and robot are stationary and the object recognition component does not need to run at a high framerate. Therefore we use SIFT together with an outlier rejection method based on clustering to identify and match features in the model and scene. The top part of Figure 5 shows a subset of the models in the database, which are matched to the scene as shown in the bottom part. The matching algorithm works as follows:

- the system takes an image of the open fridge;
- all SIFT descriptors are extracted from this image;
- for each model in the database, a kd-tree containing SIFT descriptors (created offline) is loaded;
- each descriptor in the scene is matched using the 2nd NN (nearest neighbor) test [15] against every model;
- the resultant nearest neighbors for all descriptors are clustered using a region growing approach (see below) and only the largest inlier cluster is kept;
- the model with the biggest number of inliers wins.

An unique descriptor cluster is defined as $O_i = \{\boldsymbol{p}_i \in \mathcal{P}\}$, distinct from another cluster $O_j = \{\boldsymbol{p}_j \in \mathcal{P}\}$ if:

$$\min \|\boldsymbol{p}_i - \boldsymbol{p}_j\|_2 \geq d_{th} \quad (2)$$

where $\boldsymbol{p}$ represents a descriptor point, $\mathcal{P}$ the space of all descriptors matched in the image originally, and d_{th} is a maximum imposed distance threshold. The algorithmic steps that need to be taken are:

1) create a kd-tree representation [16] for $\mathcal{P}$;
2) set up an empty list of clusters C;
3) then for every $\boldsymbol{p}_i \in \mathcal{P}$, perform the following steps:
   - add $\boldsymbol{p}_i$ to a new cluster $C_j \in C$;
   - for every $\boldsymbol{p}_i \in C_j$ do:
     - search for the set $\mathcal{P}_i^k$ of neighbors of $\boldsymbol{p}_i$ in a sphere with radius $r < d_{th}$;
     - for every neighbor $\boldsymbol{p}_i^k \in \mathcal{P}_i^k$, check if it has already been processed, and if not add it to C_j;
4) the algorithm terminates when all $\boldsymbol{p}_i \in \mathcal{P}$ have been processed and are now part of the list of clusters C.

While SIFT could be used to confidently identify the available bottles in a given high-resolution image, detection in this monocular view is insufficient for generating a high-confidence 3D grasp position. Of the PR2's two stereo camera pairs, the wide-angle stereo images were found to be too low-resolution for feature matching on the labels. Additionally, the narrow-angle stereo images did not capture enough of the scene to find all the bottles on the shelf. A faster strategy for localizing bottles than simply panning across the scene uses the following information:

- The set of available drinks, from SIFT matching in the high-resolution camera (Figure 5)
- Potential drink blobs in the 3D stereo cloud, from the wide-angle stereo (Figure 5)

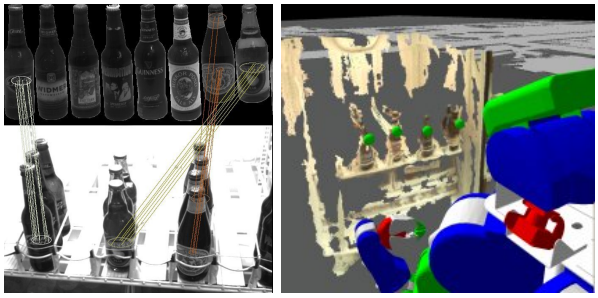

Fig. 5. Left: multiple drinks detection example using SIFT descriptors and signature clustering. The top part presents a subset of the models in the database, represented using single frontal label images, matched to the scene in the bottom part. Right: potential drinks (marked with green spheres) computed for precise detection and localization.

If the requested drink is in the set of detected drinks (from high-res SIFT), then the robot points its head at each of the potential drink blobs (from wide-angle stereo), and uses the narrow stereo system to identify whether or not each potential blob is the requested drink (with SIFT), and localize it with the narrow stereo cloud. For simplicity, we select the poses as the centroid x-y of these registered 3D points, and the closest z from the cluster with respect to the camera, with an orientation perpendicular to the largest principal component of the cluster. These grasp points are then passed through the executive (see Section VI) to the motion planner (see Section V).

D. Face Detection

Delivering a beverage requires detection of a recipient, both for reasons of perceived intelligence and safety. A robot who offers a drink to a wall or empty room has failed in its task, and thus it must detect and deliver the beverage to a person. In terms of safety, the robot must navigate around complicated obstacles with the beverage in hand, and it would be disastrous should a slight jerk in navigation cause the robot to think that a person has grabbed the bottle and release its grasp. A scenario in which the robot releases a bottle to a person who is not paying attention is equally troublesome. For these reasons, to deliver a beverage, the robot must detect that a person is present and attentive. Performing face detection of frontal faces, faces turned towards the robot, serves these requirements.

Human face detection is a well-studied problem in computer vision. The most established approach is the Viola-Jones detection algorithm, based on cascaded detectors of simple Haar-like features. The variant used in this work is following the implementation presented in [17] and available as part of the OpenCV library [18]. Even established algorithms, however, are not perfect in practice. The above detector provides an unacceptable number of false positive face detections. In our robotic application we can take advantage of the presence of depth information provided by the stereo cameras to clean up the detections. Our approach approximates a face's real-world diameter by computing the median disparity within the face's bounding box, and using the median disparity to project the bounding box into the 3D world. The final algorithm prunes face detections based on their actual 3D size, removing detections which are either too large or too small to be real faces. Detections which do not have sufficient stereo support are also removed as they were likely a result of image noise.

Fig. 6. Face detections. Our system uses depth information from stereo cameras to prune red and blue results as having unrealistic or unknown face sizes, respectively, and only acts upon green results with realistic face sizes.

In Figure 6, multiple face detection results from the basic OpenCV face detector are shown. Using stereo information, our system labels these detections as either 'red', unrealistic face size, 'blue', unknown face size, or 'green', realistic face size. The robot only acts upon green detections, thus more reliably delivering beverages.

V. Safe Arm Operation

In this application, the robot's arms and grippers are the primary tools the robot uses for physically interacting with the environment; we use the arms for opening the fridge, grasping and stowing the beverages, closing the fridge, and for unstowing and handing off the beverages. For some of these actions it is essential to use a full collision-free motion planning and execution framework; the pipeline must be used when environmental collisions are likely in the workspace where the arms are operating or where the robot is dynamically determining a goal location, such as when grasping a beverage. For the basics of the collision-free motion planning and execution framework please see previous work [12]. The framework uses sampling-based planning and collision-checking libraries to determine safe paths. When there is a reasonable expectation, however, that performing some action won't result in environmental collisions and where the goal of the arm action is known a priori, it may be acceptable to use a pre-programmed arm trajectory.

Our final approach for arm movement is associated with manipulating kinematic constrained objects like the refrigerator door. For these situations we opted to use a multi-modal local force/velocity/position control of the gripper. This is a nearly identical approach to that used in our door manipulation [11], using the *TFF* controller. Once acquiring a grasp on the door handle, the *TFF* controller is enabled, with -0.2 m/s velocity translation control normal to the door, 0 N force translation control in the other axes in the plane of the door, 0 m position control in the vertical axis, 0 N-m torque orientation control in the in the plane of the door.

In the rest of this section we focus on the particulars of our approach to safe arm operation for beverage manipulation.

A. Beverage grasping, stowing, and unstowing

Once the perception modules have produced a target grasp pose, a pre-grasp end effector position can be computed. This

pre-grasp end effector position is passed to the collision-free motion planning and execution pipeline as an end effector goal configuration. The pipeline then determines and executes a plan to achieve the goal configuration while avoiding self-collisions and collisions with occupied voxels associated with the dynamic collision map, as described in Section IV-A. Once the pre-grasp position has been achieved, the gripper is moved to a target beyond the pose estimated by the bottle localization, in order to cage the bottle and decrease the likelihood of a failure to grasp. Once in the grasp position, the robot closes its gripper and executes a pre-programmed trajectory to move the beverage.

After the beverage is in the robot's gripper, it is preferable for the robot to take paths that keep the beverage upright; turning the beverage over can shake up carbonated beverages, which will not turn into a pleasant experience upon delivery to the user. Furthermore, keeping the beverage upright will be absolutely essential for open beverages or filled glasses. Thus for stowing and unstowing we constrain the motion planner to consider only path states that largely maintain the pitch and roll of the robot's end effector. This means that the sampling-based planner will check sampled states for collision and for deviations in pitch and roll; any states that fall outside the constraint limits are rejected.

B. Beverage hand-off

The final segment of successful beverage delivery is to actually open the gripper so that the user can get the requested beverage. We wanted this component to be as safe as possible, as dropping beverages during handoff could be dangerous. As described in Section IV-D, we used face detection to ensure that a human is present in sight, before we allow the robot to consider release. Even if a face is detected, however, it still may not be safe to release the drink - we needed to make sure that the person was actually gripping the drink. The approach we took was to require that the recipient pulls on the drink. As the PR2 arm is compliant even under position control, having the recipient pull on the drink can create a sufficient spatial displacement of the end effector to enable detection and release. We employed a conservative threshold of 1 cm of displacement in the gripper's vertical axis to release; this limit was sufficient that we never released unexpectedly.

VI. Task Executive

Previously, complex closed-loop integration tests with the PR2 robots, utilized a model-based task planner and executive called *T-REX* [11] (Teleo-Reactive EXecutive). We have now began to explore multi-executive solutions to high-level robot control, under the hypothesis that different kinds of task coordination can more easily be handled with different kinds of executive frameworks. This hypothesis was motivated by the observation that the most rapid and maintainable strategies for *T-REX* model design were to describe lower-level task sequences as hierarchical finite state machines. This explicitly defines the plan that will be executed, and leaves little work to the task-planning system [19].

Our application utilized a new execution framework that lets us implement these well-defined task sequences. "Well-defined" in this case means that the nominal operation, failure modes, and the respective recovery sequences for the tasks can all be described explicitly. This new framework is a ROS-independent Python library called *SMACH* (pronounced "smash"). *SMACH* provides structures for procedurally generating programs based on hierarchical concurrent state machines. While *SMACH* is a ROS-independent library, there are several modules which allow it to seamlessly integrate with ROS constructs like messages, topics, *actionlib* actions and other systems. Developers can create a custom *SMACH* state classes that execute arbitrary Python code, but there are several parameterized state classes that make it easy to compose lower-level components and procedures when using ROS.

A. SMACH

SMACH can be used to build and execute hierarchical concurrent state machines (stateflow charts, see [20]). The basics of these design patterns have been around since the beginning of robotics [21] and have been realized with many libraries [22] [23], and in some cases even re-implemented by researches on a per-project basis. While these patterns motivate the design and structure of *SMACH*, they do not wholly define it.

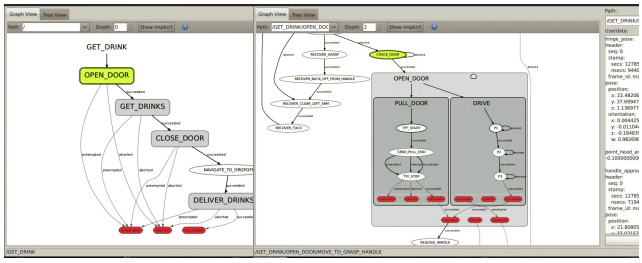

Fig. 7. Multiple levels of the *SMACH* plan for the Drinks Application as rendered by the *SMACH Viewer*. Left: The top level of the hierarchy, showing the robot actively opening the door as designated by the highlighted *OPEN_DOOR* node. Right: The mutually-preempting concurrency and state machines involved in opening the door, another level deeper into the *SMACH* plan tree.

1) Structures: There are two primary interfaces that *SMACH* defines:

- **State** - An interface for an object representing a *state of execution* with a set of potential *outcomes*. States only need to define a single `execute` function which blocks until it returns an outcome from the set of potential outcomes for that state. State *outcomes* can be thought of as the "interface" to a given state.
- **Container** - An interface for an object representing a collection of one or more *states*. Containers define different execution policies based on their child states and outcomes.

One of the simplest *SMACH* containers is the StateMachine. A *SMACH* state machine can be visualized as stateflow diagrams, where nodes are states of execution (the robot doing something) and edges represent transitions from one

state to another state via a given outcome. These containers can be composed hierarchically and have outcomes of their own.

SMACH also provides a simple split-join concurrence container. Unlike a `StateMachine`, which executes one state at a time in series, the `Concurrence` executes more than one state simultaneously. The outcomes of a `Concurrence` can be determined by one of several outcome policies defined at construction. Unless otherwise specified, a *SMACH* concurrence only terminates once all of its children have terminated.

2) User Data: Plans built with *SMACH* have capabilities that make them different from formal state machines. Each *SMACH* container has a locally-scoped dictionary of user data that can be accessed by each of its child states. If it suits the application, states can store and load data from this structure while they are executing. Like state outcomes, these inputs and outputs can be considered part of the *interface* to a state, and must be declared on construction. Similarly, containers can also have input and output keys, in order to move data between scopes.

While this feature makes analysis more complex, it also makes the system far more powerful. Not only does this allow data be passed between consecutive states, but data can be accumulated from various states in order to inform a branch later in execution. This means that the "full" state of a *SMACH* tree at any given time is the union of the active task-level states in each container and the contents of dictionary of user data in each container.

B. Application Prototyping

SMACH is designed to allow rapid and intuitive prototyping of robotics applications. The *SMACH*-ROS interface library provides a visualization interface (see Figure 7) for runtime introspection of a *SMACH* plan. The *SMACH Viewer* renders the structure of a plan, highlights the executing states at runtime, and lists the contents of the user data dictionary for a given container. This tool, along with several other *SMACH* features allowed us to quickly develop this application:

- As we discovered failure points, we could quickly add states that could estimate the nature of the failure and recover from it.
- The visualization in *SMACH Viewer* was intuitive enough to allow us to quickly find design errors through introspection, and repair the source code.
- We were able to divide the various procedures involved in our application among different people for development on different robots. We used the *actionlib* abstraction layer to encapsulate several sub-procedures, each implemented with *SMACH*.
- We developed several, increasingly capable "baseline" implementations. This required executing the application from several different starting points as each sub-procedure was improved. *SMACH*'s ability to start in the middle of a plan without changing the code can cut down on development time drastically.
- *SMACH* has built-in preemption propagation. This means that a cancel request sent to the top-level of the *SMACH* tree acts on the active state and propagates down to the leaf states in order to safely stop the currently executing tasks. This is desirable if the robot is actively manipulating a breakable object. Without this sort of software interrupt, our only options are to force-stop the executables, or press the robot's hardware run-stop, both of which might cause the gripper to release, and drop the object being manipulated.

C. Design Patterns

The task executive made use of *SMACH*'s hierarchical concurrent state machines as a baseline for structuring the plan, but within that framework made use of several other design patterns that proved useful in this application.

1) Data Coordination: Most of the *SMACH* states in our application represented *actionlib* actions, which are like preemptable service calls. These actions receive a goal message, and return some result message and status (*succeeded, aborted, preempted*). The *SMACH* user data structure allows each container to maintain a dictionary of *actionlib* goal and result messages that are relevant to the procedure as well as other data. In this application this included, but was not limited to:

- Perception feedback (available drinks, potential drink blobs)
- Composed transforms for generating grasp poses
- Drink request goals coming from the graphical user interface
- The occupancy of the drink holder on the base of the robot

2) Mutual Preemption and Topic Monitors: Concurrence containers in *SMACH* can have several termination policies defined as functions of the termination of their contained states. One previously-explored pattern [11] is mutual preemption of two concurrent tasks. This is useful when the termination of one action should cause the termination of another. In this application, a state that commands the *TFF Controller* [11] executes in parallel with a sequence of waypoints that allow it to pull the door open while simultaneously driving. Once the robot has opened the door entirely, the termination of the navigation sub-state-machine induces the preemption of the arm command.

This pattern can also be applied with other ROS-aware *SMACH* states to implement both event-driven behavior and safety mechanisms. This was done, for example, in order to preempt an actively executing arm trajectory in the event that the bottle becomes loose in the gripper.

VII. Discussions, Conclusions, and Future Work

We have evaluated the performance of our system on more than 30 trials, with different users placing up to 3 drinks orders through the web interface, and selecting a desired delivery location. A complete run (3 drinks) of the final system takes approximately 5.5 minutes under nominal execution, from the point at which the robot arrives at the fridge to the point where the fridge door is securely closed. If any of the sub-procedures fail and the robot needs to recover and retry, it can take longer.

Implementing failure recovery procedures is one of the most challenging parts of developing these applications, and *SMACH* allowed us to do this quickly and succinctly in code. While we can aim to make each section of the nominal plan execution as robust to faults as possible, errors will still happen. Many failures are simply due to incorrect perception data. Other times, some unexpected disturbance can cause the nominal plan to be interrupted. As each failure mode presented itself, we were able to quickly implement an appropriate recovery procedure in the executive. While we do not claim to have created fault-detection and recovery for all possible failures in this application, we have done so for most of the most common perception failures and manipulation failures. More importantly, the strategy we used to implement these recovery procedures can be used to augment the nominal plan without the modification of much code in the executive.

To avoid failures in the perception system, we minimized the number of false positives in drink detection and localization by applying more conservative acceptance thresholds. False negatives during localization only cause the task executive to catch the failure and trigger the appropriate recovery actions (retry, in this case). One problem that could cause drink localization to fail, was the absence of visual features on the surface of bottles that were rotated with their labels facing away from the cameras. This is a pathological problem in the current system, but could be resolved by investigating more active sensing strategies, where bottles with few features could be rotated by the robot until identified.

Detection of the fridge handle requires a "good" orientation of the robot, since the reflective coating on the fridge inhibits poor stereo perception performance. However, while this detection pose is not robust to large variations in lighting and orientation, if the detection fails, or gives a false positive, and the robot does not achieve a grasp on the door handle, it will return to the detection pose and retry.

There also are several procedures that were implemented open-loop in the interest of time that could benefit from additional sensor data in the future. Placing bottles on and unloading bottles from the base, for example, is performed with pre-recorded trajectories. The PR2s forearm cameras have proved useful for performing close-proximity sensing, and they could be used to verify occupancy of the drink holder as well as localization of drinks on the base.

We have presented a system for identifying, fetching, and delivering drinks with the PR2 autonomous robot. Our contributions include the development and integration of several modules for perception, motion planning, and system task executive. All the software components have been integrated as part of the ROS infrastructure and are available for download as open source packages at `http://www.ros.org/wiki/Papers/ICRA2011_Drinks`.

Though we believe we have made important steps towards identifying the key issues that need to be solved towards building autonomous robotic assistants, the problem is **not** solved yet, and many open issues still remain. Scaling such a system for handling larger varieties of refrigerators and drinks, etc, are still problems that we need to work on.

REFERENCES

[1] I. (Information, R. T. F. to Support Man, and A. S. at University of Tokyo, "Mission Statement," Tech. Rep., 2007.

[2] C. C. Consortium, "A Roadmap for US Robotics: From Internet to Robotics," Tech. Rep., May 21 2009.

[3] E. R. T. Platform, "Robotic Visions To 2020 And Beyond – The Strategic Research Agenda For Robotics in Europe," Tech. Rep., June 2009.

[4] U. Hillenbrand, B. Brunner, C. Borst, and G. Hirzinger, "The Robutler: a Vision-Controlled Hand-Arm System for Manipulating Bottles and Glasses," in *35th International Symposium on Robotics*, 2004.

[5] U. Reiser, C. Connette, J. Fischer, J. Kubacki, A. Bubeck, F. Weisshardt, T. Jacobs, C. Parlitz, M. Hägele, and A. Verl, "Care-o-bot®3: creating a product vision for service robot applications by integrating design and technology," in *IROS'09: Proceedings of the 2009 IEEE/RSJ international conference on Intelligent robots and systems*. Piscataway, NJ, USA: IEEE Press, 2009, pp. 1992–1998.

[6] S. Srinivasa, D. Ferguson, C. Helfrich, D. Berenson, A. Collet, R. Diankov, G. Gallagher, G. Hollinger, J. Kuffner, and M. VandeWeghe, "Herb: A home exploring robotic butler," 2009.

[7] M. Scheutz, P. Schermerhorn, C. Middendorff, J. Kramer, D. Anderson, and A. Dingler, "Toward affective cognitive robots for human-robot interaction," in *In AAAI 2005 Robot Workshop*, 2005, pp. 9–13.

[8] T. Asfour, K. Regenstein, P. Azad, J. Schröde, and R. Dillmann, "Armar-iii: A humanoid platform for perception-action integration." in *2nd International Workshop on Human-Centered Robotic Systems (HCRS'06)*, 2006.

[9] E. Neo, T. Sakaguchi, K. Yokoi, Y. Kawai, and K. Maruyama, "A behavior level operation system for humanoid robots," dec. 2006, pp. 327 –332.

[10] E. Marder-Eppstein, E. Berger, T. Foote, B. Gerkey, and K. Konolige, "The Office Marathon: Robust Navigation in an Indoor Office Environment," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, Anchorage, Alaska, May 3-8 2010.

[11] W. Meeussen, M. Wise, S. Glaser, S. Chitta, C. McGann, P. Mihelich, E. Marder-Eppstein, M. Muja, V. Eruhimov, T. Foote, J. Hsu, R. Rusu, B. Marthi, G. Bradski, K. Konolige, B. Gerkey, and E. Berger, "Autonomous Door Opening and Plugging In with a Personal Robot," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, Anchorage, Alaska, May 3-8 2010.

[12] R. B. Rusu, I. A. Sucan, B. Gerkey, S. Chitta, M. Beetz, and L. E. Kavraki, "Real-time Perception-Guided Motion Planning for a Personal Robot," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, October 11-15 2009.

[13] R. B. Rusu, W. Meeussen, S. Chitta, and M. Beetz, "Laser-based Perception for Door and Handle Identification," in *International Conference on Advanced Robotics (ICAR)*, June 22-26 2009.

[14] R. B. R. et al, "PCL (Point Cloud Library)," Tech. Rep., 2009 – 2010. [Online]. Available: http://www.ros.org/wiki/pcl

[15] D. G. Lowe, "Distinctive Image Features from Scale-Invariant Keypoints," *Int. J. Comput. Vision*, vol. 60, no. 2, pp. 91–110, 2004.

[16] M. Muja and D. G. Lowe, "Fast approximate nearest neighbors with automatic algorithm configuration," *VISAPP*, 2009.

[17] R. Lienhart and J. Maydt, "An extended set of haar-like features for rapid object detection," in *ICIP*, 2002.

[18] "OpenCV," http://opencv.willowgarage.com/wiki/.

[19] C. McGann, E. Berger, J. Bohren, S. Chitta, B. P. Gerkey, S. Glaser, B. Marthi, W. Meeussen, T. Pratkanis, E. Marder-Eppstein, and M. Wise, "Model-based, hierarchical control of a mobile manipulation platform," in *ICAPS Workshop on Planning and Plan Execution for Real-World Systems*, Thessaloniki, Greece, 2009.

[20] D. Harel, "Statecharts: A visual formalism for complex systems," 1987.

[21] N. J. Nilsson, "Hierarchical robot planning and execution system," Stanford Research Institute, Tech. Rep., April 1973.

[22] P. J. Lucas, "An object-oriented language system for implementing concurrent, hierarchical, finite state machines," Master's thesis, University of Illinois at Urbana-Champaign, Urbana, Illinois, 1993.

[23] B. Lee and E. A. Lee, "Hierarchical concurrent finite state machines in ptolemy," *Application of Concurrency to System Design, International Conference on*, vol. 0, p. 34, 1998.