BakeBot: Baking Cookies with the PR2

Mario Bollini, Jennifer Barry, and Daniela Rus

***Abstract*— We present BakeBot, a PR2 robot system that bakes cookies autonomously, from *mise en place* presentation through baking in a toaster oven. The baking planning and control system is implemented as a hierarchical finite state machine. We developed parameterized motion primitives for baking. The motion primitives utilize the existing sensing and manipulation capabilities of the PR2 platform and also our new compliant control techniques to address environmental uncertainty. The system was tested through 27 baking attempts, 16 of which successfully resulted in edible cookies.**

I. Introduction

The creation of a robot chef is a grand challenge for robotics. Accomplishing this requires advances in the fields of computer vision, task and motion planning, and manipulation. We present BakeBot, a first step towards a robot able to execute any recipe. BakeBot consists of a suite of planning and control algorithms for the Willow Garage PR2 robot that enable it to bake cookies autonomously, from *mise en place* ingredient presentation through baking.

In this paper we describe a robot system for autonomous baking. We assume the robot is presented with a set of ingredients placed in bowls, much like we would expect in a real kitchen. The robot is also given the sequence in which the ingredients should be mixed together. The goal of the robot is to mix the ingredients in the correct order, and place the resulting dough in a baking tray in the oven. The robot does not know the exact location of the ingredients nor the mixing bowl. It uses its stereo cameras and laser scanner to locate all of the bowls on the table. Given the bowl locations and the sequence of mixing operations, the robot (1) plans paths to add the ingredients to the mixing bowl and (2) generates compliant mixing trajectories for folding in the ingredients.[1] Finally, the robot places the dough in a tray and places it in an oven for baking. After 20 minutes, the robot removes the baked dough from the oven.

The kitchen provides a rich array of tasks that range from pick and place through manipulation of semi-solid materials such as doughs and batters. The environment is semi-structured, allowing for many reasonable simplifying assumptions to be made while transitioning a robot control program from the block world or a simulation space. The inherent organization of a kitchen around a human-centric workspace, the relative similarity of kitchen tools and supplies to their peers (most spoons look similar, most bowls look similar, etc.), and the uniformity of kitchen tasks such as mixing, pouring, and placing into an oven provide an opportunity to apply the latest object recognition and motion planning theory in an environment that is familiar and controlled, but able to provide additional challenges as the field progresses.

The main technical challenges in this work have been (1) locating the ingredients, (2) planning a motion path to add a new ingredient to the mix, (3) using two hands to mix the ingredients into a uniform dough, and (4) building an end-to-end system. The system in this paper was tested extensively in the context of making chocolate Afghan cookies following an Australian recipe [1]. The ingredients are butter, sugar, flour, cocoa, and rice krispies. These ingredients provide a wide range of textures and consistencies for testing the scope of the adding and mixing modules. This paper's contributions are (1) an end-to-end system for robot baking, (2) an algorithm for locating ingredients on a broad tabletop, and (3) a compliant motion algorithm capable of mixing ingredients of different texture into a uniform dough inside a bowl.

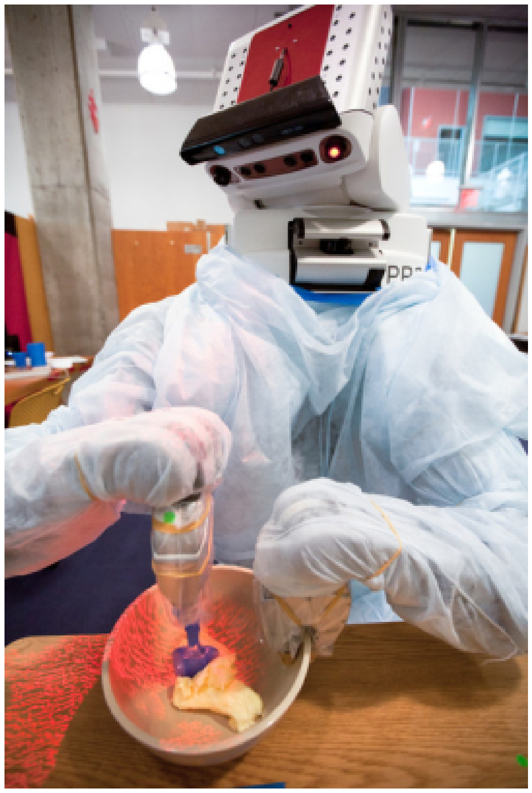


Fig. 1. The PR2 in the protective garmet. The garmet mitigates the risk of spills and splashes during the cooking process.

All authors are with the Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, MA, 02139. {mbollini, jbarry, rus}@csail.mit.edu

[1]Different ingredients have different textures and require different patterns of mixing.

A. Related Work

BakeBot utilizes the low-level manipulation and perception system developed in Rusu et al. (2009) [2]. It combines the individual components of the tabletop manipulation system to execute a complicated household task, following the roadmap outlined in Kemp et al. (2007) [3]. Unlike the pancake duo discussed in Beetz et al. (2011) [4], the BakeBot system is built from the bottom up, with the initial focus on developing and refining the motion primitives rather than on the high-level task planner.

Although our current work focuses on the lower-level motion primitives, we are designing the subtasks used by BakeBot to be easy to write in the language of symbolic planners. Hybrid symbolic and geometric planners have had success in recent years in integrating high-level task planning with lower-level geometric motion planning [5]–[7].

II. System Setup

A. Hardware

BakeBot uses the Willow Garage PR2 humanoid manipulation platform. The platform utilizes an omnidirectional wheeled base and two 7-degree of freedom (DOF) torque controlled arms to achieve an active workspace similar to that of an average adult human. The PR2 has two single DOF grippers for object manipulation. The sensor suite on the robot includes two pairs of stereo cameras, a tilting planar laser range finder, and a high resolution camera. The system is controlled by a pair of onboard computers, one of which controls the actuators over a gigabit-LAN network in a 1kHz real-time loop.

The PR2 is modified with the attachment of a spatula to the right gripper. The spatula is rigidly attached to the gripper with bolts, eliminating the opening degree of freedom. This side-steps the need to grasp the spatula with the PR2 and to maintain a sufficient power grasp to control the spatula during aggressive mixing motions. The PR2 is covered in water-resistant surgical gowns that have been modified to fit the robot. These gowns, shown in Figure 1, protect the robot from splashes and particles. The left gripper is covered in waterproof plastic held in place with rubber bands, which also help grab the bowls (as the fingertip rubber pads are covered with plastic).

B. Software

BakeBot is implemented as a hierarchical finite state machine using the `SMACH` ROS package. The state machine ran as a single ROS node on the robot. The top-level machine delegated to separate state machines for bowl collection, mixing, scraping, and putting the cookie sheet into the oven. These mid-level machines then delegate to more specific state machines when necessary to encapsulate complex subtasks, like cleaning the spoon or rotating the mixing bowl. The lowest level of control is accomplished by client classes that controlled the actual robot in the current process through the ROS messaging and service systems. Other ROS nodes were implemented to handle the mixing, scraping, and logging in separate processes. These are separated from the main program for robustness and ease of development.

The ROS `Pick and Place Manager` [2] was heavily modified and is used as the basic manipulation and detection framework. The primary modifications were focused on increasing the robustness of the inverse kinematic path planning and execution and on extending the detection workspace across the tabletop. Smaller modifications were made to enable the system to preserve and reserve state when used by multiple program classes and when the system was restarted. Considerable work was done to create a system that could be restarted at arbitrary points, allowing for quick debugging and recovery from failures.

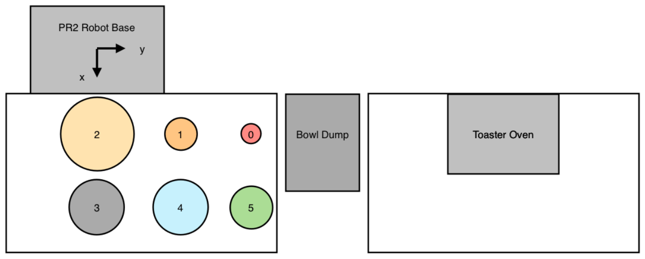


Fig. 2. The environment setup for the baking task. The absolute positions of the bowls on the table surface is not enforced as long as a minimum distance between the objects is maintained. Bowl 0 contains the cocoa, 1 the sugar, 2 is the mixing bowl, 3 is the cookie sheet, 4 contains the rice krispies, and 5 the flour.

C. Environment Setup

The cookies are created in a kitchen environment consisting of two work surfaces, one for preparation and another to support a standard toaster oven (preheated to 320^oF). Figure 2 shows the kitchen layout. Both tables are immobile during the test, and constant distances are maintained between the robot's starting location, the preparation surface, and the oven. This eliminates the need for localization and navigation, neither of which is the focus of this project. Such capability has been demonstrated on the PR2 and we intend to include it in future generations of the system.

Six items are placed on the preparation table at the start of the baking process: four plastic ingredient bowls of various sizes and colors, a large plastic mixing bowl, and a metallic pie pan. The ingredients (butter, sugar, cocoa, flour, and rice krispies) are pre-measured and distributed among the bowls on the table. The softened butter is placed in the mixing bowl, primarily to mitigate the risk of spills during manipulation, while the solid ingredients are distributed among the other bowls. The metal pie pan is greased with butter to prevent the cookies from sticking. Pre-measuring the ingredients and arranging them on the work surface dramatically simplifies the task of baking with a robot and is a reasonable simplifying step as it is commonly done for human chefs. The items are arranged in a 2 x 3 grid on the table, with the relative position of every item the same during each run. The absolute positions of the objects are not enforced, though a minimum separation is maintained between the objects.

III. Baking Algorithm

The baking task is broken into subtasks with the intent of creating a library of parameterized motion primitives that can be applied to other recipes and different kitchen tasks. These subtasks are:

1) Locating the ingredients on the table
2) Collecting all of the ingredients into the mixing bowl
3) Mixing everything together into a homogeneous batter
4) Pouring the mixture into the pie pan
5) Locating the oven handle and opening the oven
6) Placing the cookie sheet into the oven

Figure 3 is a pictoral timeline of the robot executing these subtasks.

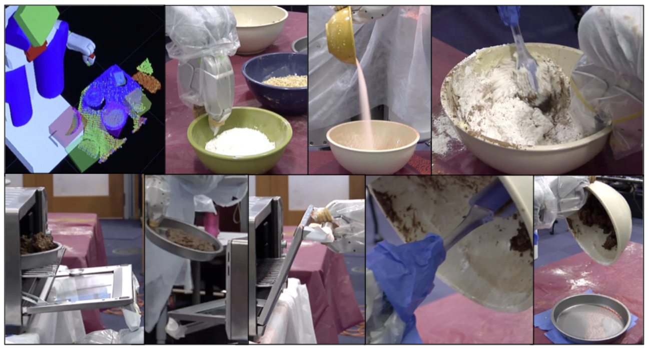


Fig. 3. A pictoral timeline of the baking process, clockwise from top left: object recognition, grabbing the ingredients, pouring ingredient into mixing bowl, mixing the ingredients, pouring the batter into the pie pan, scraping the mixing bowl, opening the oven, placing pan into oven, closing oven.

A. Generating an action plan

We assume the robot is given the high-level motion plan for baking, and that the plan is expressed as a sequence of robot baking actions from our robot's repertoire. There are many planners [5]–[7] that can be used to generate such a sequence. Thus we start with a finite state machine composed of individual states representing motion primitives that are parameterized (with environmental and task-specific parameters) at runtime. For example, a grasp action is a motion primitive. Neither the object to grasp nor the inverse kinematic solutions for the grasp are known a priori. At runtime, the object detection system locates the object on the table and the grasp solution is computed. The state machine was assembled to execute the recipe for chocolate Afghan cookies.

B. Object recognition

We utilize the ROS tabletop manipulation pipeline [2] to detect the bowls on the table surface. The ingredients are identified by their relative position on the table. We experimented with HSB thresholding to differentiate between ingredients and found that it was successful distinguishing the colored bowls and ingredients (such as the sugar from the cocoa) from one another, but was not adequate for distinguishing between textures (such as between the flour and the sugar). We modified the tabletop manipulation package with

```
Data: Objects on the table surface $T$, detected objects $D$,
      minimum object separation $r_{min}$, object list $K$
Result: $K^*$, the list of detected objects on $T$
Divide the surface $T$ into a grid $G$;
for every gridpoint $g$ on $G$ do
    tabletop detection centered at $g$;
    for every detected object $d$ do
        for every object $k$ in $K$ do
            calculate the distance $r$ between $d$ and $k$;
            if $r$ is greater than $r_{min}$ then
                add $d$ to $K$;
            else
                calculate the volumes $v_d$ and $v_k$;
                if $v_d$ is less than $v_k$ then
                    remove $k$ from $K$ and add $d$ to $K$;
                end
            end
        end
    end
end
for every object $k$ in $K$ do
    tabletop detection centered at $k$;
    add object closest to midpoint of $k$ to $K^*$
end
```
Algorithm 1: Broad Tabletop Detection

Algorithm 1 to allow us to detect and grasp objects across the entire tabletop surface. This algorithm combines the results of many tabletop detections across the table surface using an object distance heuristic. It eliminates duplicates and provides software handles to grab any of the detected objects using the tabletop manipulation grasping utilities.

C. Collecting ingredients together

Collecting ingredients into the mixing bowl is accomplished by breaking the subtask into a sequence of motions for grasping, lifting, horizontal carrying (to avoid spilling the contents of a bowl), and pouring. Algorithm 2 outlines the procedure for collection of an ingredient. We utilize the tabletop manipulation package for grasp planning and the OMPL planner to find collision-aware inverse kinematic trajectories that satisfied constraints in our Cartesian workspace. Ingredient collection was implemented as a hierarchical finite state machine within the overall system architecture. This compartmentalizes individual actions into reusable primitives. The primary advantage of such an arrangement is that it allows us to use a simple set of decision points to determine whether it is necessary to replan or to choose a different goal pose entirely.

The task space for pouring the ingredients into the mixing bowl is discretized into four pour orientations, each with its own pre- and post-pour poses. If planning or execution of the inverse kinematic path to one of these poses fails, the state machine determined whether to replan to a nearby pose (within some delta of the original desired pose) or to replan to another of the discretized poses. The state machine also allows us to easily incorporate base movement into motion planning without adding to the dimensionality of the search space used to find an inverse kinematic solution. If a grasp or motion fails to find an inverse kinematic solution and the goal pose is outside of a nominal subset of the overall workspace, the base is moved to bring the goal pose into that region. After the subtask is complete (such as completing a pour), the base is moved to its original position.

By breaking the ingredient collection operation into fine-grained components a simple finite state machine is used to generate complex and robust behaviors. Creating states that have uniform entry and exit conditions allows the state machine to be easily rearranged without conflicting with the requirements of other actions later in the baking process.

D. *Mixing*

Mixing is the most manipulation intensive component of the system. It combines joint position control, planning inverse kinematic paths for Cartesian trajectories, and force-compliant control of the end effector.

Algorithm 3 describes the mixing process. The PR2 constrains the position of the mixing bowl by executing a grasp on the left of the bowl and maintaining it during the mixing motion. The grasp pose is chosen to maximize the mixing workspace. To execute the grasp, the PR2 plans a path placing the left manipulator directly over the left-most rim of the bowl. The manipulator is then lowered to the height of the rim of the bowl and closed. This is more effective than regular grasp planning for two reasons: the bowl is very close to the robot in an already highly constrained workspace, and the contents of the bowl are irregular and sometimes present graspable features. This method guarantees exact repeatability of the maneuver.

The mixing was performed using the `ee_cart_imped_controller`[2], which allows for the execution of force/impedance trajectories in the Cartesian workspace. Mixing trajectories are generated at runtime based on the detected position of the mixing bowl.

Several different kinds of mixing trajectories are implemented to achieve adequate mixing. The first is a circular mixing trajectory that scrapes the spatula around the inner perimeter of the mixing bowl. The trajectory is a circular position/stiffness trajectory of radius $0.10m$ larger than the diameter of the mixing bowl and located $0.05m$ above the rim of the mixing bowl. The controller stabilizes the center of the end effector, not the active tip of the spatula, to the trajectory. Figure 5(a) shows the path of the center of the end effector during circular mixing and Figure III-D shows the joint torques during circular mixing. At every step of the trajectory the spatula is moved further around the circumference of the bowl and is pushed down into the bowl.

The stiffnesses around the positions on the trajectory were chosen to keep the spatula against the sides of the bowl during the mixing process. The controller bandwidth, backlash in the arm, and the density of points on the generated trajectory cause the end effector to jitter during the mixing action. This helps keep the spatula from getting stuck in the batter.

Circular mixing tends to force the batter, which has an approximate consistency of Play Doh, to accumulate

[2]a ROS package released by Barry, Bollini, and Hiu

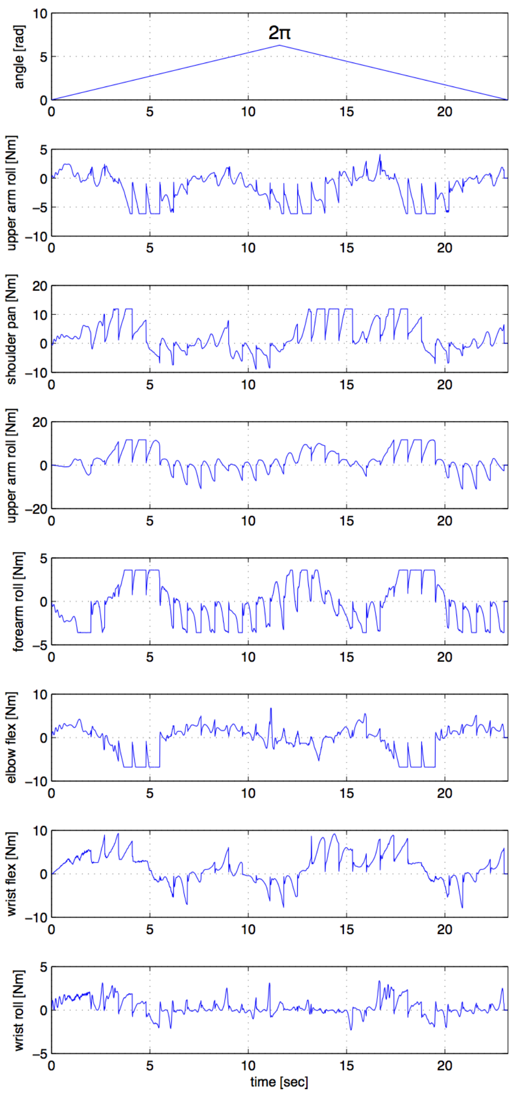


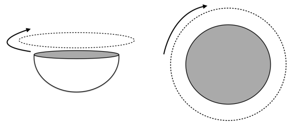
Fig. 4. Joint torques during execution of the circular mixing trajectory in a mixing bowl full of cookie batter. The spoon angle is plotted in the top plot, the trajectory moves the spatula around the bowl once, then reverses its direction and ends with the spatula in the starting position.

against the manipulator that constrains the bowl. Periodically switching the direction of circular mixing trajectory helps reduce the size of the batter buildup but ultimately causes smaller amounts of it it to be pressed against both sides of the manipulator. This is dealt with by switching the sides of the bowl held by the manipulator. By releasing the grip on the mixing bowl, switching the grip across the bowl to the other side, and translating and rotating the bowl so that the manipulator is back on its left side, the buildup can be moved to an area of the inner circumference that allows it to be more easily broken up by the circular mixing trajectory.

The second mixing trajectory is composed of linear motions through the center of mixing bowl. This linear mixing is accomplished by generating a series of waypoints alternately inside and outside of the bowl. These waypoints are strung together and interpolated, creating a trajectory that moves the spoon back and forth through the center of the bowl. Figure 5(b) shows the path of the center of the end effector during linear mixing. This position/impedance trajectory moves the batter around and helps to even out any irregularities produced by the circular mixing.

At the end of every mixing cycle the spatula is cleaned by scraping it against the lip of the mixing bowl, usually dislodging a considerable amount of stuck batter. This action is accomplished by executing a force trajectory pushing it down and across the rim of the bowl.

The mixing is performed open-loop with respect to the dough consistency and homogeneity, executing a pattern of compliant motions that guarantee that the ingredients will be thoroughly combined.


(a) Circular mixing trajectory.

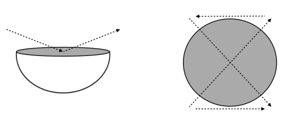

(b) Linear mixing trajectory.

Fig. 5. Top and side views of the compliant trajectories followed by the center of the end effector during circular 5(a) and linear 5(b) mixing routines.

E. Pouring the final mixture

Pouring the final mixture from the mixing bowl into the pie pan utilizes two motion primitives: the pouring primitive, parameterized for the mixing bowl and the pie pan, and the mixing primitive, parameterized for a horizontal scraping motion on the upturned mixing bowl. Balancing the workspaces of the compliant controller on the right arm (with the mixing spoon) and the joint position controller on the left arm (holding the upturned bowl over the pie pan) proved difficult and required considerable trial and error to find a solution that worked consistently. The result of this process is one giant cookie in the pie pan.

F. Placing the cookie sheet into the oven

The robot drives from the table to the oven by following an open loop path. The oven handle is located using cloud segmentation in front of the combined surface of the oven door and the hanging tablecloth. The robot localizes itself with this detection and modifies its position to maximize the probability of opening success. Once it has achieved a grasp on the oven handle, a force-compliant trajectory is generated to pull the handle backwards and down.

Once the oven is opened, the robot moves to the table, grasps the pie pan, and returns to the oven. The pie pan grasp is executed using the same motion primitives that were used to grasp the side of the mixing bowl, parameterized for the pie pan. The orientation of the end effector relative to the pie pan while it is on the table is recorded. This is used as the reference orientation for a horizontal pie pan, ensuring robustness to variations in the angle of the pie pan caused by unmodeled interactions between the lip of the pan and the gripper pads. The pie pan is inserted into the oven by planning and executing a path through a series of precomputed Cartesian waypoints while maintaining the horizontal pose of the pie pan. The oven is closed by executing a joint-space path with the left gripper, slowly impacting the oven door and forcing it to close. The oven is reopened after twenty minutes have elapsed.

```
Data: Ingredient I, Mixing Bowl B
Result: The PR2 pours the contents of the bowl I into the
        mixing bowl B
perform a tabletop detection to refine positions of I and B;
plan a grasp on B;
if grasp could not be planned then
    move robot base to bring I within nominal grasp region;
    plan a grasp on B;
end
plan collision-free path to move I above B for known
orientation k in known list K do
    rotate to k;
    if rotation could not be planned then
        continue;
    else
        rotate I into pouring configuration for orientation k if
        rotation could not be planned then
            continue;
        else
            shake I;
            break;
        end
    end
end
move I to place position P;
if move could not be planned then
    translate robot base closer to P;
    move I to place position P;
end
place I;
translate robot base to starting position;
```
Algorithm 2: ADD INGREDIENT

IV. Experimental Data

Each subsystem of BakeBot was tested more than 100 times. The full end-to-end system was tested 27 times. The goal of these experiments has been to establish the robustness of the end-to-end BakeBot system and to collect performance data for each operation, including time and feedback from the robot's sensors, such as the torques in Figure III-D. Figure 6 shows these tests plotted horizontally across the task space. The tests are plotted chronologically, from top to bottom. Circles on the figure represent failures that required human intervention to correct. Minor failures, such as the fingers slipping off of the oven door halfway through the opening procedures, were corrected during runtime and the tests were allowed to continue. More serious failures that required the system to be fully restarted or a new piece of code to be written caused the termination of the test. In all, 16 tests ran to completion, with an average runtime of 142 minutes from start to finish. The plurality of the runtime was spent mixing the ingredients. Each of the two mixing actions takes, on average, 27 minutes to execute. Each of the four ingredient collections takes 8 minutes to execute. Table I shows the average runtime for each of the subtasks.

TABLE I
AVERAGE RUNTIMES FOR EACH OF THE BAKING SUBTASKS.

Subtask	Avg. Duration [sec]	Executions per cookie batch	Avg. Total Time [sec]
Broad tabletop detection	118	1	118
Adding an ingredient	477	4	1908
Mixing	1610	2	3220
Scraping onto cookie sheet	978	1	978
Putting into oven	1105	1	1105

```
Data: Mixing bowl $B$
Result: The PR2 mixes the contents of the $B$ together.
Perform a tabletop detection to refine position of $B$;
move base to bring $B$ into nominal mixing region;
grasp left side of $B$ with the left end effector;
move spatula over center of mixing bowl in collision-free way;
move spatula down into mixing bowl;
execute compliant mixing trajectories;
move spatula to original pose in collision-free way ;
release mixing bowl;
return base and end effectors to their original poses;
Perform a tabletop detection to refine position of $B$;
```
Algorithm 3: MIX

The BakeBot system was created by assembling and supplementing stock PR2 manipulation and perception packages. The majority of the failures outlined in Figure 6 reflect the shortcomings of this approach, as the added robustness in our hierarchical state machine was insufficient to address the occasionally inconsistent behavior of the base PR2 software packages in an environment more complex and uncertain than that for which they were designed. For example, the tabletop manipulation package occasionally chose to grasp the ball of batter in the center of the mixing bowl, rather than the rim of the bowl, because the batter presented a more salient grasp feature than the rim of the bowl. While this failure only occurred twice during testing, the large number of environmental interactions and sequential subtasks to complete the baking task make BakeBot particularly sensitive to even low probabilities of subsystem failure. Given the complexity of the baking task and the stock manipulation and perception capabilities of the PR2 platform, we feel that BakeBot represents a successful demonstration of a first step towards the creation of a more general robot chef. Future implementations will have to address uncertainty at a lower level of the overall PR2 system in order to provide more robust performance than was demonstrated by BakeBot.

V. DISCUSSION

Our goal is to create a robot chef that is able to execute recipes in standard kitchens. BakeBot represents our early progress towards this goal, demonstrating that a task-based hierarchical state machine combined with basic planning and manipulation are sufficient to accomplish the complicated task of baking cookies. The more general application of the project is twofold: BakeBot demonstrates a use of compliant and force control to compensate for uncertainty in the environment and BakeBot uses a set of hierarchical subtasks that could be modified and re-ordered to achieve many different types of household goals.

In the process of making cookies, BakeBot must work with many different objects and surfaces in the environment. Modeling each of these surfaces is an infeasible task, so for many motion primitives, such as mixing and scraping, we use compliant control algorithms that do not require models. For example, we use a force trajectory for the task of scraping the spatula against the side of the mixing bowl. Instead of trying to precisely model the rim of the bowl and scraping along it using a complicated position-controlled trajectory, we use a force trajectory that directs the spatula downwards and back towards the robot. The result is that the spatula is lowered to, and scraped along, the rim of the bowl without ever using precise knowledge about the position or shape of the rim. BakeBot uses this strategy effectively in multiple motion primitives, increasing its versatility and robustness as these trajectories require very little a priori knowledge about the environment and adapt well to uncertainty in the environment.

BakeBot also uses hybrid position and force trajectories to bypass the need for complicated planning in constrained situations. For example, in opening the oven, BakeBot uses position control in one dimension and force control in the other. The result is a robust opening trajectory that is indifferent to the positioning of the oven relative to the robot and the robot's grasp of the oven handle. Unlike most algorithms for opening a door, this trajectory requires no planning time and no information about the oven except the width of its door.

The world of BakeBot is non-deterministic and there are many possible failure modes for each primitive. By designing a set of hierarchical subtasks and identifying these failure modes, we have a versatile set of subplans that could be used with many different task-level symbolic planners for many different tasks. For example, the oven-opening subtask can be re-parameterized for cupboard opening, while the ingredient-collecting task could just as easily be used to put bowls away. There are many such kitchen tasks that could be done with only re-parameterizations and re-orderings of the tasks used in BakeBot, and many more that would require the introduction of only a small number of extra tasks.

At the moment, adapting the BakeBot subroutines to a new kitchen task would require creating another finite state machine specific to the new goal. However, as we have described, the subtasks are intrinsically hierarchical, each building on sets of lower-level subtasks and primitives, and, although non-deterministic, each has a set of indentifiable failure modes, as well as methods for determining success. Moreover, each subtask has a symbolic set of pre-conditions

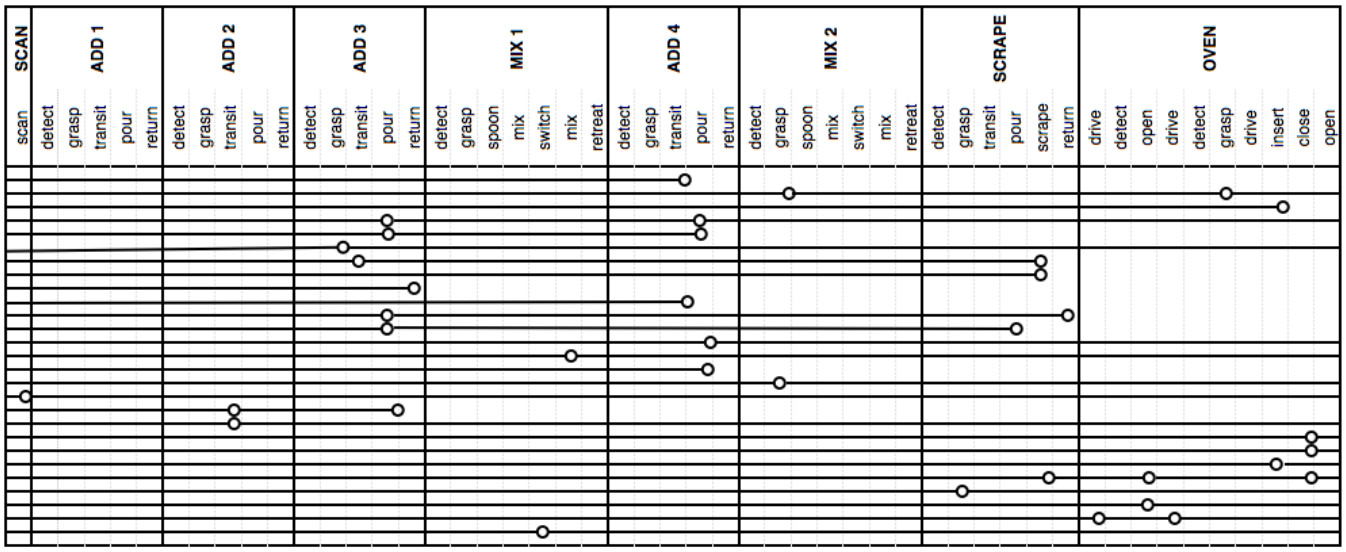


Fig. 6. This chart shows the failure modes of the baking attempts. Trials are plotted chronologically from top to bottom. A trial starts at the left with tabletop detection and ideally complete at the right with an oven opening. Circles represent failures that required human intervention. Trials that end in a circle represent a failure that could not be quickly recovered from, while other trials were able to recover with slight human interaction and continue.

and effects. Therefore, these subtasks can be easily integrated into a hybrid symbolic and geometric planner, such as a hierarchical task network [7], [9] or the hierarchical planning in the now (HPN) framework [6].

Integrating task-level planning into BakeBot has several advantages. First, a task-level planner can order subtasks to achieve any goal; by writing the subtasks as an HTN or in the HPN framework, we can use the same planner to bake cookies or clean the kitchen. Moreover, in a hard-coded finite state machine every outcome of a subtask must be predicted and handled; most task-level planners remove this difficulty by planning in the "now". Rather than attempting to predict every possible outcome, they make vague top-level plans and then carry out as much of that plan as possible, re-planning on the fly if the system reaches an un-planned for state. Using the knowledge about what needs to be accomplished, symbolic planners can reason how to get there from the *current* state. By not requiring a full policy for every possible state, the symbolic planners will allow us to represent the baking task in a more compact manner.

Although BakeBot does not currently use task-level reasoning, we have provided a library of useful, hierarchical, and symbolically-represented subtasks that could be integrated into task-level planning. In future work, we plan to use the HPN framework to plan for cookie baking and kitchen cleaning using these subtasks.

VI. Conclusion

BakeBot, the cookie-baking robot, is an encouraging initial realization of a robot chef. Implementing and combining the tasks of ingredient collection, mixing, scraping, and oven-opening required creating a set of robust, hierarchical subtasks with recoverable failure modes. We used a wide variety of algorithms to ensure a high success probability for each subtask, including lower-level finite state machines, stereo perception, and hybrid force/position trajectories. As a result, BakeBot was able to complete 16 cookie-baking runs, requiring minimal human intervention despite a running time of over two hours and fifty sequential subtasks.

VII. Acknowledgments

The authors would like to acknowledge Anne Holladay for her work on opening the oven. We also thank the team at Willow Garage for the opportunity to work with their robot platform through the PR2 beta program. This work was supported in part by an NDSEG Fellowship and a grant from the NSF.

References

[1] "Afghan biscuits," http://australianfood.about.com/od/bakingdesserts/r/AfghanBiscuits.htm.

[2] R. Rusu, I. Sucan, B. Gerkey, S. Chitta, M. Beetz, and L. Kavraki, "Real-time perception-guided motion planning for a personal robot," in *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on.* IEEE, 2009, pp. 4245–4252.

[3] C. Kemp, A. Edsinger, and E. Torres-Jara, "Challenges for robot manipulation in human environments [grand challenges of robotics]," *Robotics & Automation Magazine, IEEE*, vol. 14, no. 1, pp. 20–29, 2007.

[4] M. Beetz, U. Klank, I. Kresse, A. Maldonado, L. Mösenlechner, D. Pangercic, T. Rühr, and M. Tenorth, "Robotic Roommates Making Pancakes," in *IEEE-RAS International Conference on Humanoid Robots*, 2011.

[5] F. Gravot, S. Cambon, and R. Alami, "aSyMov: A Planner that Deals with Intricate Symbolic and Geometric Problems," in *International Symposium on Robotics Research*, 2003, pp. 100–110.

[6] L. P. Kaelbling and T. Lozano-Pérez, "Hierarchical Planning in the Now," in *IEEE Conference on Robotics and Automation*, May 2011.

[7] J. Wolfe, B. Marthi, and S. Russell, "Combined Task and Motion Planning for Mobile Manipulation," in *Interational Conference on Automated Planning and Scheduling*, 2010.

[8] J. Barry, M. Bollini, A. Holladay, L. P. Kaelbling, and T. Lozano-Pérez, "Planning and Control Under Uncertainty for the PR2," in *IROS PR2 Workshop*, September 2011, Extended Abstract.

[9] D. Nau, T. C. Au, O. Ilghami, U. Kuter, W. J. M. D. Wu, and F. Yaman, "SHOP2: An HTN Planning System," *International Journal of Artificial Intelligence*, vol. 20, pp. 379–404, 2003.