Interpreting and Executing Recipes with a Cooking Robot

Mario Bollini, Stefanie Tellex, Tyler Thompson, Nicholas Roy, and Daniela Rus

1 Motivation

The creation of a robot chef represents a grand challenge for the field of robotics. Cooking is one of the most important activities that takes place in the home, and a robotic chef capable of following arbitrary recipes would have many applications in both household and industrial environments. The kitchen environment is a semi-structured proving ground for algorithms in robotics. It provides many computational challenges, such as accurately perceiving ingredients in cluttered environments, manipulating objects, and engaging in complex activities such as mixing and chopping. Yet it also allows for reasonable simplifying assumptions due to the inherent organization of a kitchen around a human-centric workspace, the consistency of kitchen tools and tasks, and the ordered nature of recipes. We envision a robotic chef, the BakeBot, which can collect recipes online, parse them into a sequence of low-level actions, and execute them for the benefit of its human partners. We present first steps towards this vision, by combining techniques for object perception, manipulation, and language understanding to develop a novel end-to-end robot system able to follow simple recipes and by experimentally assessing the performance of these approaches in the kitchen domain.[1]

2 Problem Statement

This paper describes progress towards a robotic system which is able to read and execute simple recipes. The robot is initialized with a set of ingredients laid out on the table and a set of natural language instructions describing how to use those ingre-

Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, e-mail: {mbollini, stefie10, tcthompson, nickroy, rus}@csail.mit.edu

[1] Video of recipe execution: http://people.csail.mit.edu/mbollini/bakebot_planning.mp4

dients to cook a dish such as cookies, salad, or meatloaf. For example, ingredients might include flour, sugar, butter, and eggs arrayed in labeled bowls on the table, and instructions might include statements like "Add sugar and beat to a cream." We assume the robot has access to a model of the environment including the set of ingredients and their labels, and the location of tools such as the oven. In addition, it has a repertoire of primitive actions, such as pouring ingredients into a bowl and mixing them. The robot must parse the text of recipe and infer an action sequence in the external world corresponding to the instructions. It then executes the inferred action sequence on the PR2 robotic manipulation platform, performing the motion and task planning necessary to follow the recipe to create the appropriate dish. For example, for the instructions above, the robot might empty the bowl of sugar into the mixing bowl, then stir the ingredients in the bowl. Figure 2 shows the BakeBot system inputs and outputs.

Fig. 1 BakeBot, a PR2 robot system that executes simple baking recipes. The PR2 is modified with a protective covering to prevent damage from spills and a spatula which is bolted to the end effector to enable mixing.

3 Related Work

Many previous authors have described robotic systems for following natural language instructions. Previous work focused on understanding natural language route directions [9, 10, 6] and mobile-manipulation commands [12]. Similar to existing approaches, our system learns a model that maps between the text of the instructions and a reward function over actions in the predefined state-action space, specifically focusing on the domain of following recipes. Recipes present a unique challenge

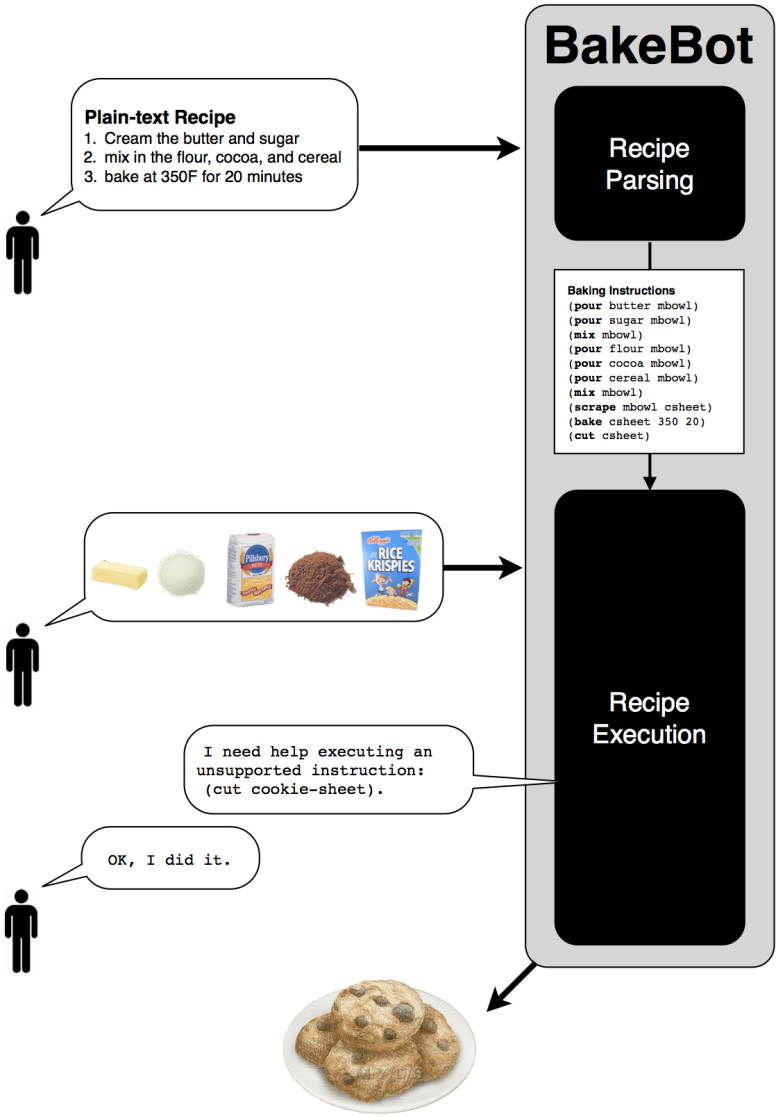

Fig. 2 The human interaction with the BakeBot system for recipe execution. First the person provides the plain-text recipe and the measured ingredients. Then BakeBot infers a sequence of baking primitives to execute that correspond to following the recipe. If BakeBot encounters an unsupported baking primitive, it asks its human partner for help executing the instruction. The end result is baked cookies.

to these approaches because of the richer space of actions inherent to the cooking domain. BakeBot uses the low-level manipulation and perception system described in Rusu et al. [11]. Beetz et al. [2] have demonstrated dispensing pancake batter from a premixed container and flipping the pancakes on a skillet. In this paper we demonstrate an end-to-end robot cooking system capable of implementing any baking recipe that requires pouring, mixing, and oven operations on premeasured ingredients provided to the system. Our system is able to follow recipes downloaded from the internet; we demonstrate it by following two different recipes in the real world and by further evaluating its performance on a larger test set in simulation.

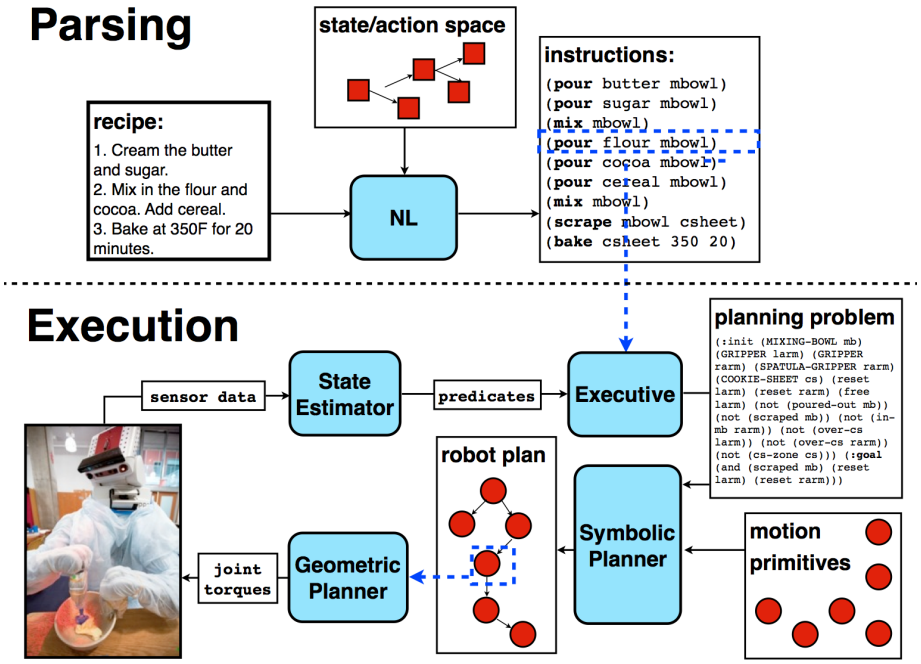

Fig. 3 Architecture of the BakeBot system. The NL system processes the plain text recipe, producing a high-level plan which is sent to the robot. For each instruction in the high-level plan, the motion planner assembles a motion plan and executes it on the PR2 robot.

4 Technical Approach

The robot's goal is to read the text of a natural language recipe, and use it to infer an action sequence in the environment that corresponds to preparing the dish described in the recipe. The robot first segments the recipe into sentences based on punctuation. Then for each sentence, it infers an action sequence in the environment

corresponding to the words in the sentence. After executing the action sequence successfully, the robot will have produced the appropriate dish, for example cookies, brownies, or meatloaf. The system architecture appears in Figure 3.

We formally define a state-action space for the kitchen domain using primitive actions such as *mix*, *pour*, and *bake*. Within this state space, many specific action trajectories can be followed, yielding a variety of different dishes. In order to follow a specific recipe, we define a reward function based on the text of the recipe. The robot uses forward search to find the sequence of states and actions that maximizes reward, then executes those actions in the external world. Because the text of the recipe generates the reward function, this optimization corresponds to finding an action sequence that follows the recipe.

4.1 State/Action Space for the Kitchen

We define a state S_k as the collection of unused ingredients S_k^i in the workspace, the mixing bowl S_k^b and its contents, the cookie sheet S_k^s and its contents, and a toaster oven S_k^o and its temperature and contents. Given any state S_k, we define $actions(S_k)$ to be the set of available actions in that state:

- For each unpoured ingredient S_k^i, $pour(S_k^i, S_k^b) \in actions(S_k)$.
- If $nonempty(S_k^b)$ then $mix(S_k^b) \in actions(S_k)$.
- If $nonempty(S_k^b)$ then $scrape(S_k^b, S_k^s) \in actions(S_k)$.
- If $empty(S_k^o)$ then $preheat(S_k^o) \in actions(S_k)$.
- If $empty(S_k^o) \wedge nonempty(S_k^s)$ then $bake(S_k^s) \in actions(S_k)$.

After executing an action such as *pour* in state S_k, the next state S_{k+1} contains one less unused ingredient, and the mixing bowl S_{k+1}^b contains the corresponding poured ingredient. Although the currently available actions do not always execute robustly on the real robot, they support a surprising array of recipes, ranging from brownies to meatloaf to salads. Over time we plan to increase the robustness of the existing actions and add additional ones such as chopping and whisking in order to increase the range of recipes that the robot supports.

If no supported action exists for an instruction, the robot prompts the user, visually via the terminal and audibly via a text-to-speech synthesizer, to execute the action. This enables a wider array of actions than the manipulation platform and motion primitive set currently support by leveraging support from a human partner.

4.2 Reward Function

In order to follow a recipe, the system takes as input the natural language text of a recipe and induces a reward function over the state/action space. The reward function is learned from a labeled dataset of recipes paired with the correct action se-

quence. Formally, a recipe consists of a list of sentences $d_1 \ldots d_N$. For each sentence, the system infers a sequence of states $S_1 \ldots S_K$ that maximizes the reward function R:

$$\underset{S_1 \ldots S_K}{\operatorname{argmax}} \sum_j R(d_j, S_m \ldots S_n) \tag{1}$$

We define the reward function as a probability distribution parametrized using a log-linear model [3]. The model is trained from a corpus of recipes annotated with the correct action. The feature functions are bag-of-words features crossed with the presence of specific actions and arguments in the state sequence. For example, a feature would be "Mix" $\in d_j \wedge pour \in S_1 \ldots S_k$. The system takes positive examples from the annotations. To form negative examples, it finds action trajectories which result in an incorrect final state. The system then finds model parameters which maximize the likelihood of this training set using a gradient descent algorithm.

We use a beam search to jointly optimize the action sequence across multiple sentences. The sequence of states and actions is then passed to the execution model to run on the physical robot.

4.3 Plan Assembly

The inferred state/action sequence is interpreted into a robot action plan of *baking primitives*. Each baking primitive corresponds to a single cooking action and consists of a collection of motion primitives with STRIPS-style preconditions and effects and a STRIPS-style description of the goal[8]. The action plan is represented as a sequential finite state machine of ordered baking primitives corresponding to the inferred state/action sequence.

If no supported baking primitive exists for an action in the state/action sequence a *user interaction* state is created and inserted into the state machine, enabling the system to follow recipes that might involve instructions for which no baking primitives exist. For example, if the system encounters a primitive for chopping walnuts, which it cannot execute itself, it asks its human partner for assistance (via text output and audibly via a text to speech synthesizer).

4.4 Plan Execution

Once a sequence of baking primitives has been inferred, the robot executes them sequentially. When executing a baking primitive, the system (1) infers a set of Boolean planning predicates that correspond to the world state, (2) symbolically plans a sequence of motion primitives to execute the baking primitive from the current state, (3) parametrizes the motion primitive sequence into an executable sequential state machine, and (4) executes the state machine. If execution fails the system estimates

the current state and replans a new sequence of motion primitives to accomplish the baking primitive. This enables the system to recover from many errors experienced during recipe execution.

4.4.1 Detecting and Pouring Ingredients

We utilize the ROS tabletop manipulation pipeline [11] to find the ingredients on the preparation table. First the system finds objects on the table using an algorithm [5] that combines multiple detections across the tabletop into a single state representation of the objects in front of the robot. The system is given a mapping between segmented objects and ingredient names; no object recognition is used to match the clusters to an object database. This approach provides more robust performance to bowl size variation, and more importantly, is more robust to varying ingredient volumes (and shapes) within the ingredient and mixing bowls.

Once the robot has detected ingredients on the table, it manipulates them using the *pour* baking primitive. The *pour* baking primitive contains the following motion primitives: ingredient bowl grasps, lifting, moving an object over the table with orientation constraints, pouring, shaking, and discarding an empty ingredient bowl. The task space for bowl grasps and pours are discretized into four cardinal bowl orientations at the 12, 3, 6, and 9 o'clock positions around the bowl circumference. This simplified task and motion-plan specification limits bowl positions to a small, intuitive set while providing versatility. If planning or pouring from a bowl grasp pose fails another pose is attempted. After the ingredient is poured, the robot discards the ingredient bowl.

4.4.2 Mixing and Scraping

The *mix* and *scrape* baking primitives use motion primitives for bowl grasps, moving an object over the table with orientation constraints, pouring, moving the spatula across the table, mixing, and scraping. These primitives use both end effector position and compliance control techniques.

For mixing, the robot performs a grasp of the mixing bowl to hold it in place. It moves the other end effector, with an attached spatula, via end effector position control over the mixing bowl. It then activates the end effector Cartesian force/compliant controller [1], plunging the spatula down into the batter. Next the robot executes a series of linear and circular mixing trajectories to ensure the batter is well-combined. Mixing is performed open-loop: the system does not know how well-mixed the batter is. We attempted to use joint-torques to estimate batter consistency but this was unsuccessful. Other approaches include using computer vision techniques to visually sense the batter consistency. The mixing strategy aims for well-combined batter, at the expense of time and the risk of over-mixing.[2] When

[2] Overmixing a flour-based batter can cause baked goods to become tough.

mixing is completed the system uses a compliant trajectory to wipe the spatula on the rim of the mixing bowl before switching back to position control to reset the end effector.

To initiate scraping, the robot grasps the mixing bowl and lifts it over the cookie sheet. It pours the contents of the bowl into the cookie sheet, then moves spatula inside of the bowl. Next it executes a compliant scrape trajectory to dislodge any batter that remains stuck inside of the mixing bowl. This trajectory also presses the batter down into the cookie sheet, ensuring one large flat cookie.

4.4.3 Oven Operation

The *bake* baking primitive uses motion primitives for base navigation, handle recognition, grasping, following endpoint trajectories, and compliant oven opening.

The robot first drives an open-loop path to the oven and servos to its final position based on the detection of the oven handle. It locates the handle using point cloud segmentation in front of the combined surface of the front of the oven and the hanging tablecloth under the oven. Localization on the oven handle eliminates the need for mapping as the odometry of the PR2 base is sufficient to drive from the preparation table to within range to perform oven handle detection. The robot opens the oven by grasping the handle and executing a hybrid force/compliant trajectory. The trajectory follows position waypoints pulling away from the oven while maintaining a constant downward force, opening the oven door with limited *a priori* information about the oven [4].

The robot drives back to the table, grasps the cookie sheet, and returns to the oven. It inserts the cookie sheet into the oven by following a path of end-effector waypoints in the workspace. Finally it closes the oven with the same manipulator by following a joint trajectory that brings the end effector into contact with the open door and pushes the door upwards. After baking is complete, the robot follows a similar process to remove the finished product from the oven.

5 Experiments

We evaluate our approach in three ways. First, we performed two experiments using our platform: we passed recipes through our language processing system to create robot instruction sets, and we executed the instruction sets on the physical robot system. The robot's workspace is initialized with a set of objects containing the ingredients and implements necessary to follow the recipe; the system does not parse the ingredients list. Finally, we compared our recipes to human performance by comparing the number of baking primitives supported by our framework to those needed for a set of ten recipes sampled from Joy of Cooking.

We collected a dataset of 60 recipes from the internet, describing how to make simple dishes such as brownies, meat loaf, and peach cobbler. For each recipe, we

Recipe Text	**Inferred Action Sequence**
Afghan Biscuits	
200g (7 oz) butter	
75g (3 oz) sugar	
175g (6 oz) flour	
25g (1 oz) cocoa powder	
50g cornflakes (or crushed weetbix)	
Soften butter.	$pour(butter, bowl); mix(bowl)$
Add sugar and beat to a cream.	$pour(sugar, bowl); mix(bowl)$
Add flour and cocoa.	$pour(flour, bowl); pour(cocoa, bowl)$
Add cornflakes last.	$pour(cornflakes, bowl); mix(bowl)$
Put spoonfuls on a greased oven tray.	$scrape()$
Bake about 15 minutes at 180°C (350°F).	$preheat(350); bake(pan, 20)$

Fig. 4 Text from a recipe in our dataset, paired with the inferred action sequence for the robot.

formally specified the specific ingredients and implements necessary to follow the recipe. This initialization is given to the robot. Next, for each instruction in the recipe text, we annotated the sequence of primitive actions the robot should take in order to follow that instruction. We used 45 recipes from this corpus to train the model, and 15 recipes to test it. A sample recipe from the test set appears in Figure 4, together with the automatically inferred action sequence.

5.1 Real-world Demonstration

The robot system operates in a kitchen environment consisting of two work surfaces, one for preparation and another to support a standard toaster oven (preheated to the temperature specified in the instruction set).

We assume that the kitchen is *mise en place*; ingredients are pre-measured and distributed on bowls on the table. Equipment includes four plastic ingredient bowls of various sizes and colors containing premeasured ingredients, a large plastic mixing bowl, and a metallic pie pan. The items are arranged in a grid on the table, with the relative position of every item noted in our ingredient-resolution program.

First, we assessed the robustness of the physical capabilities of the robot by performing extensive tests on a single recipe: "Afghan Biscuits" (which appears in Figure 4). Minor failures, such as the fingers slipping off of the oven door halfway through the opening procedures or the inverse kinematic planner requiring a restart, were corrected during runtime and the tests were allowed to continue. More serious failures, such as spilling an ingredient or scraping the contents of the mixing bowl onto the floor or table, that required the system to be fully restarted or a new piece of code to be written caused the termination of the test. We tested the full end-to-end system, from robot instructions through finished dish, 27 times on this recipe,

of which 16 ran to completion, with an average runtime of 142 minutes from start to finish. On average, mixing ingredients in the bowl takes 27 minutes to execute. Adding an ingredient takes on average 8 minutes; scraping from the bowl onto a cookie sheet takes 16 minutes, and putting an item into the oven takes roughly 18 minutes. The robot is able to successfully execute actions in a cooking domain, although much more slowly and with many more failures than a human. A pictorial representation of the baking process is shown in Figure 5.

Next, we selected two recipes from our test set and used our recipe following system to execute them from plain-text recipes through the finished dish. We executed two runs each of "Afghan Biscuits" and "Quick'N Easy Sugar Cookies," demonstrating the end-to-end system integration and the robot's ability to physically execute several recipes.

5.2 Quantitative Evaluation of Recipe Processing

We performed a quantitative evaluation in simulation to assess the system's ability at finding a plan to understand a variety of recipes. First, we assessed the system's performance at inferring an action sequence for specific sentences in the recipes. The 15 recipes in the test set contained a total of 92 individual sentences. Of these sentences, the system correctly inferred exactly correct plans for 49% of them. Many of the errors consist of omitting an ingredient, as in "Mix with hands – form small balls and flatten on ungreased pan." which referred to all ingredients in the recipe; for this command the system inferred only $pour(sugar, bowl); mix(bowl)$ but excluded other ingredients such as butter and flour.

Next, we evaluated the system's performance when inferring the complete action sequence for the recipes. The system correctly inferred action sequences for 26.67% of the test set (4/15). As in the instruction-level tests, most of the errors consist of omitting an ingredient for a single instruction, resulting in the rest of the action sequence becoming incorrect.

Finally, we assessed the performance of instruction-level and end-to-end inference with respect to changing training corpus size in Figure 6 and Figure 7. We kept the recipe test set constant through these tests. We started with a training set of one random recipe, trained the system with that training set, and evaluated the performance of the system using the test set. At each step, we added 5 random recipes (up to a maximum of 45 recipes) to the training set. The results for the instruction-level evaluations are shown in Figure 6 and the results of the end-to-end evaluations are shown in Figure 7. These graphs reflect the asymptotic nature of the training set size; we see a large initial improvement in inference performance when we add more recipes to the training set, but the improvement diminishes with each addition step.

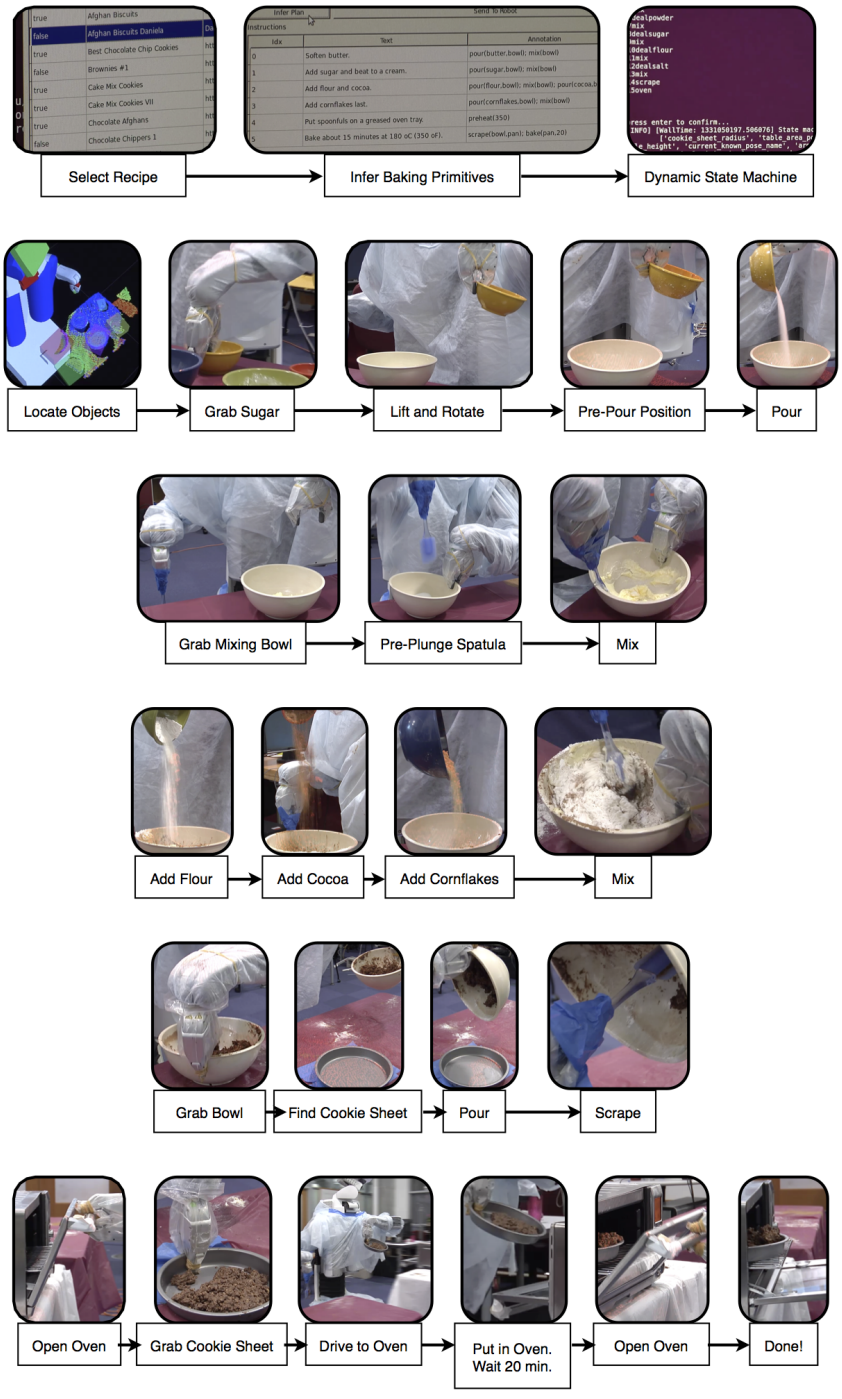

Fig. 5 A pictorial timeline of the baking process.

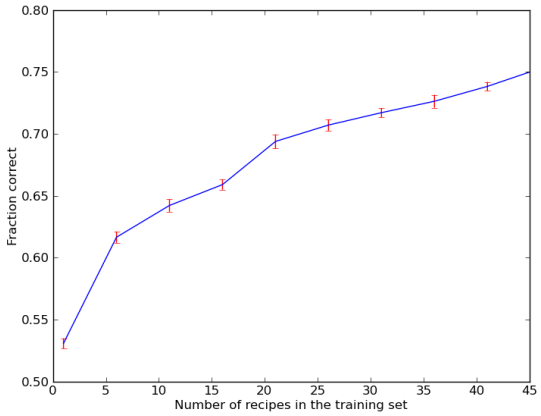

Fig. 6 Instruction-level recipe inference performance with varying training set size.

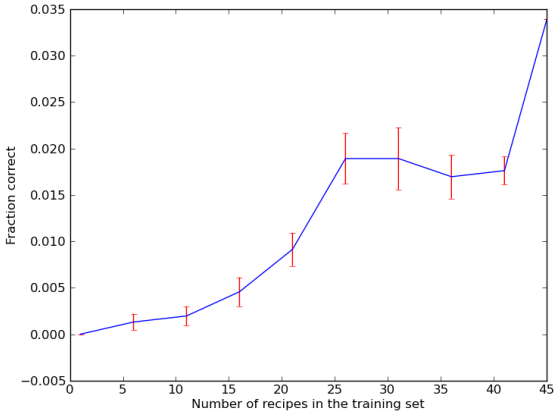

Fig. 7 End-to-end recipe inference performance with varying training set size.

5.3 Comparison to Human Performance

Finally, we assess the types of actions supported by the system compared to a human chef. First, we compared our supported primitives to a brownie recipe performed by a human, annotated within the CMU Multimodal Activity Database [7]. We found that our *mise en place* assumption eliminated 67% of the annotated actions. Of the remaining actions performed by the human, our instruction set supports 79%. The two unsupported types of actions were greasing a cookie sheet and cracking eggs. Our compliant controller could be used to implement a greasing technique (using

Crisco and a paper-towel, for example). However, cracking eggs is unsupported because of limitations in the PR2's manipulation and perception capabilities.

Next, we compared our instruction set to a set of ten recipes sampled from Joy of Cooking [13], ranging from corn dogs to salsa fresca.[3] The system is able to execute 67% of the 120 instructions from these recipes. Adding motion primitives to support waiting for specific events, such as boiling or tenderness, as well as stove operations would increase the fraction of supported instructions considerably. It is worth noting that while the BakeBot system supports a large subset of the recipe instructions, it lacks the dexterity and perception to perform more complicated actions. Any aspiring chef can attest that simply going through the motions of a recipe is insufficient to guarantee tasty food. For example, when making scones, an instruction like "Cut in [cold butter]... until the size of small peas" is challenging even for human chefs to execute so that the scones have their distinctive flakiness.

We compare these figures to our corpus of simple baking recipes, of which 76% were supported by the BakeBot system. Of the 101 unsupported sentences, 24 were waiting for the cookies/brownies to cool before removing from the pan, 17 were associated with greasing the cookie sheet, and 8 were simply descriptive statements or friendly notes (i.e. "Do not overbake" and "Enjoy!").

While performance can be increased by adding more manipulation primitives, we believe that improved system extereoception would yield the most significant recipe execution benefits. Many recipes in the Joy of Cooking set and within our baking recipe corpus require feedback to determine whether or not a step of the recipe is completed. For example, a recipe may require onions to be sauted until translucent. First order implementations to sense this onion state may use vision systems to identify color transformations. More capable systems may use thermal imaging to sense the temperature of the onions and correlate this temperature to the desired activation temperature of the chemical reaction associated with onion translucence. We see the long-term potential for robotic cooking systems to leapfrog human chefs through the integration of advanced sensing and feedback systems.

6 Main Experimental Insights

In this paper we presented BakeBot, a robotic chef that is capable of following textual recipes in a kitchen environment. We demonstrated that the robot is able to successfully follow two different recipes in the real world. In addition, we performed a quantitative evaluation of the system's performance at following instructions from a larger set of recipes in simulation. Our approach represents first steps towards a robotic chef capable of executing arbitrary recipes in the home environment.

Executing the recipes on the robot shows a functioning end-to-end system that successfully integrates language processing, object recognition, task planning, and

[3] Recipes from Joy of Cooking: corn dogs, tart green salad, eggs in a basket, fettuccini with butter and cheese, Israeli couscous pilaf, carrot cake, classic scones, salsa fresca, Belgian beef stew, and roasted whole salmon.

manipulation. Our set of motion primitives enable a variety of simple baking recipes to be executed on the robot, and we demonstrated two successfully experimentally. The compartmentalization of motion planning and execution into primitives enables the system to support a wide array of recipes while minimizing software development and debugging effort.

The natural language component of the system shows promising performance at understanding individual instructions; it inferred exactly correct actions for many instructions, and many of the mistakes were due to missing one or two ingredients from aggregate phrases such as "Sift together the dry ingredients." However, the robot's physical and perceptual capabilities limited the scope of the recipe processing. For example, the system ignores a warning such as "Do not brown; do not over bake" because detecting the color of overcooked cookies through an oven door is an unsolved perceptual problem. BakeBot is a demonstration of the challenges faced when integrating large robotic control systems. The overall robustness of the BakeBot is limited by the inherent complexity of a system executing such a difficult task; the top-level cooking system is vulnerable to faults in its software substrate. The difficulty of maintaining a system built upon a constantly updated software base reflects a need for improved software management for robotic systems.

The key limitation of the system is its lack of robustness. Failure in any of the robot's systems leads to a failure to successfully follow the recipe: in the natural language system, the perceptual system, or the control system. Actions in the real-world demonstration often failed to execute successfully because of inability to adapt to unexpected situations in the environment. For example, flour being piled too high in a bowl caused the grasp planner to aim for the flour rather than the edge of the bowl. For another 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. The large number of environmental interactions and sequential baking primitives to complete the baking task make BakeBot particularly sensitive to even low probabilities of subsystem failure.

Our experimental performance was demonstrative of the issues faced in the construction of large robotic systems. The performance of complex tasks such as cooking require the interaction of many systems and subsystems. This iteration can often be error-prone and difficult to fully characterize or debug, oftentimes leaving the top-level system vulnerable to faults in its software substrate.

We have focused thus far on creating a functioning end-to-end system for semi-structured home environments. System robustness will improve as its underlying manipulation software matures and as better perception of the environment increases the system's ability to deal with uncertainty and correct its own mistakes. We plan to expand the repertoire of the system by designing additional motion primitives to enable actions such as chopping, blending, and frying, introducing new challenges for manipulation, perception, and natural language understanding.

References

[1] Jennifer Barry, Mario Bollini, and Huan Liu. End effector Cartesian controller, 2012. URL http://www.ros.org/wiki/ee_cart_imped.

[2] Michael Beetz, Ulrich Klank, Ingo Kresse, Alexis Maldonado, Lorenz Mösenlechner, Dejan Pangercic, Thomas Rühr, and Moritz Tenorth. Robotic roommates making pancakes. In *IEEE-RAS International Conference on Humanoid Robots*, 2011.

[3] Adam L. Berger, Stephen A. Della Pietra, and Vincent J. Della Pietra. A maximum entropy approach to natural language processing. *Computational Linguistics*, 22:39–71, 1996.

[4] Mario Bollini. Following recipes with a cooking robot. Master's thesis, MIT, 2012.

[5] Mario Bollini, Jennifer Barry, and Daniela Rus. Bakebot: Baking cookies with the PR2. In *International Conference on Intelligent Robots and Systems (IROS)*, Vilamoura, Portugal, Oct. 2012. (in submission).

[6] David L. Chen and Raymond J. Mooney. Learning to interpret natural language navigation instructions from observations. In *Proc. AAAI*, 2011.

[7] Fernando De la Torre Frade, Jessica K Hodgins, Adam W Bargteil, Xavier Martin Artal, Justin C Macey, Alexandre Collado I Castells, and Josep Beltran. Guide to the Carnegie Mellon University multimodal activity (CMU-MMAC) database. Technical Report CMU-RI-TR-08-22, Robotics Institute, Pittsburgh, PA, April 2008.

[8] Jörg Hoffmann and Bernhard Nebel. The FF planning system: Fast plan generation through heuristic search. 14:253–302, 2001.

[9] Thomas Kollar, Stefanie Tellex, Deb Roy, and Nicholas Roy. Toward understanding natural language directions. In *Proc. ACM/IEEE Int'l Conf. on Human-Robot Interaction (HRI)*, pages 259–266, 2010.

[10] Cynthia Matuszek, Dieter Fox, and Karl Koscher. Following directions using statistical machine translation. In *Proc. ACM/IEEE Int'l Conf. on Human-Robot Interaction (HRI)*, pages 251–258, 2010.

[11] 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 *Intelligent Robots and Systems, 2009.*, pages 4245–4252. IEEE, 2009.

[12] S. Tellex, T. Kollar, S. Dickerson, M. R. Walter, A. G. Banerjee, S. Teller, and N. Roy. Approaching the symbol grounding problem with probabilistic graphical models. *AI Magazine*, 32(4):64–76, 2011.

[13] I. von Starkloff Rombauer, M.R. Becker, E. Becker, and M. Guarnaschelli. *Joy of Cooking*. Scribner Book Company, 1997.