Crust & Conquer: A pizza-making robot for your everyday needs*

*Note: Final project made for 6.4212: Robotics Manipulation - Fall 2024

Matthew Gardner
Massachusetts Institute of Technology
Cambridge, USA
mattg156@mit.edu

Marine Maisonneuve
Massachusetts Institute of Technology
Cambridge, USA
mmaisonn@mit.edu

***Abstract*—Crust and Conquer presents a solution for autonomously assembling ingredients onto a pizza using a mobile KUKA IIWA arm and a depth camera for perception. The project presents a mobile manipulator capable of selecting and distributing ingredients over pizza dough, achieving a 14% coverage area in 28 seconds. To achieve this, our team implemented a dual-controller state machine, a perception pipeline leveraging convex hull and pixel clustering algorithms, and dynamic transitions between differential and global inverse kinematics. While the system met its ingredient coverage target, challenges with hardware setup, environment modeling, and realistic grasping dynamics limited further functionality. Future work includes extending the state machine to incorporate spreading multiple ingredients, refining perception with improved ingredient detection and object segmentation methods, and scaling the solution for simultaneous pizza production. This project highlights the potential of robotic systems in food preparation and the complexities involved in achieving real-world applicability. Our code can be found here.**

***Index Terms*—pizza, tomato, pizza oven, Nicholas, hunger, iiwa**

I. Introduction

As graduate students, few comforts compare to pizza during midterms and project weeks. This project aims to automate pizza making, with broader implications for the food industry. Robotic systems in food preparation can reduce foodborne illness risks, improve quality and accessibility, and enhance worker conditions by eliminating late-night shifts. While current robotic solutions lack the speed and cost efficiency of human workers, advancements in this area could address immediate student needs and create long-term benefits for public health, food access, and labor in the food service industry.

From a technical perspective, this project presents an intriguing robotics challenge as we aim to improve the speed and efficiency of robotic "spreading" techniques. Currently, most robotic applications in the food industry rely on specialized tools, such as tongs [12], or handle ingredients one at a time [14]. In this final project, we are exploring an approach that mimics more "human-like" motions. Specifically, we hypothesize that an iiwa robot could grab a bowl full of ingredients, tilt it over the pizza dough, and shake it to distribute the ingredients evenly.

II. Related Work

We separate this section is 4 main categories of related work: Mobile IIWA, Perception, State Machine, and General. These resources helped us define our problem as well as provided insights in some of the hurdles of the project.

A. Mobile IIWA

- [1] This chapter of the class textbook, titled Mobile Manipulation, provided key insights in the planning and kinematics of a mobile iiwa (iiwa with an x,y,z mobile base).
- [2] This exercise from the Mobile Manipulation chapter mentioned above was instrumental to our mobile iiwa set up. It provided guidance to set up our IK controller with the mobile iiwa and which ports to link to have the robot move.
- [3] RobotLibrarian showed a good use of mobile iiwa and the inverse kinematics controller, and it informed our use of "pizza state estimation" to determine the ingredient coverage. For example, it shows a good perception pipeline: point cloud segmentation, object ICP, and Gripper Alignment.

B. Perception

- [4] This article explains the convex hull, Andrew's Monotone Chain Algorithm, and how to implement it. This was key to enable our perception timeline, as we were able to estimate the total area of each ingredient from our point cloud using this method.
- [5] This is an article detailing density-based partial clustering of applications with noise (DBSCAN).
- [6] [8] Both resources are chapters of the Robotics Manipulation textbook: Pose Estimation and Segmentation. These were used as resources regarding point cloud usage.
- [7] This notebook was used specifically as a resource to hook up the camera to our system.

C. State Machine

- [9] The exercise portion of this notebook provided context into state machine definition in robotic manipulation.
- [10] We decided early on to switch between two controllers differential inverse kinematics and inverse kinematics to better execute complex, arbitrary trajectories. To do this, we needed to implement a state machine that switched between both of them. Nicholas' example was key to our implementation and provided an example implementation of a dual controller state machine.
- [11] The clutter clearing notebook provided DiffIK logic running in a state machine and provided fixes for some key hardware issues we were having during DiffIk controller resets.

D. General

- [12] Process Flow and Culinary Station Set Up reference. This is an example of a robot arm in a typical culinary station setup and steps the robot has to take to execute a delivery. This is insightful regarding the amount of time it takes to make a food item due to specific pick and placing.
- [13] News article on similar pizza robot concept. This provides some business insight on public interest in robotics applications in the service industry.
- [14] Drake environment setup & conceptual parent to pizza robot. Our workstation setup is directly inspired from this project.
- [15] Drake C++ Documentation.

III. Technical Approach & Methods

A. Hardware Setup

A full representation of our Diagram can be found here. Hardware setup for the project relies on the mobile_iiwa implementation from the (manipulation course files. The mobile_iiwa implementation adds three free "joints" for the XYZ translational position of the base. An InverseDynamics controller accepts a the generalized robot state, a twenty-wide vector of positions and velocities. Since all controls and trajectories are produced in position space, we connect a StateInterpolatorWithDiscreteDerivative block to the input. This block converts position commands to the generalized robot state. A SchunkWsg gripper is welded to the last iiwa link (link 7) and controlled by a SchunkWsgPositionController block. This gripper controller implements a force limit set to 500 N to aid grasping of a heavy bowl and ingredients while minimizing slip.

The hardware components of our perception system are taken from several examples in the manipulation course notes. Our scene uses a generic RGBD camera placed in the world above the table. Using a DepthImagetoPointCloud system, we connect an RGB image, Depth image, and point cloud to our Planner for ingredient coverage ("pizza state") estimation.

To execute both small precise motions (shaking the bowl to spread ingredients) and large mobile-base motions (moving between different workstations), our system includes dual position control schemes. For local proximity operations, we execute trajectories with a DifferentialInverseKinematicsIntegrator ("DiffIK") controller which produces smooth motions minimizing the change in joint position for as many joints as possible. Additionally, our Planner implements a global IK solver with piecewise polynomial interpolation in joint space to produce substantially improved iiwa robot poses and trajectories when traveling between the two different tables. The two position controllers are swapped using a PortSwitch connected to the robot's StateInterpolatorWithDiscreteDerivative. To better control the mobile iiwa arm's behavior, we also connected a custom DiffIK ParameterUpdater system to our Planner; this allows our robot to lock and unlock individual axes of the mobile base on the fly during execution.

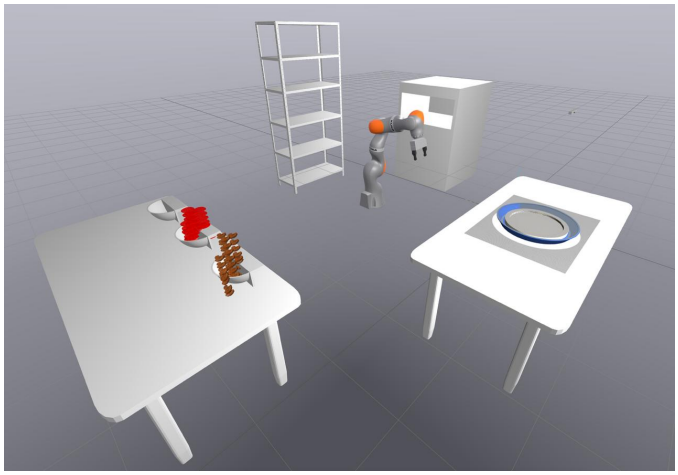


Fig. 1: Environment Setup. Our scene contains 1 pizza, 3 ingredient bowls (two filled with tomatoes and mushrooms, one awaiting sausages), one pizza oven, and one delivery shelf for pizzas awaiting pickup. At center is a mobile_iiwa arm with onboard gripper control and planner. A camera is fixed to the environment above the pizza.

B. Environment Setup

Our environment set-up includes pizza assembly, ingredients, oven, and delivery areas as seen.

The pizza assembly is composed of a table with a camera centered above it and a pizza pan with a pizza dough. The pizza pan is centered on the table and directly below the camera. This allows the camera to see the state of the pizza as it is being assembled and monitor the pizza states. The ingredients area is another table in the scene with 3 bowls on the tabletop. Two of those bowls have ingredients in them, one having mushrooms and the other having tomatoes. The ingredients and bowls are free falling objects in the scene, so the robot can freely pick up or put down the bowl and the ingredients can freely slide out of the bowl. The bowl has a convex mesh. The oven and delivery area are both convex shapes with enough space for a pizza pan.

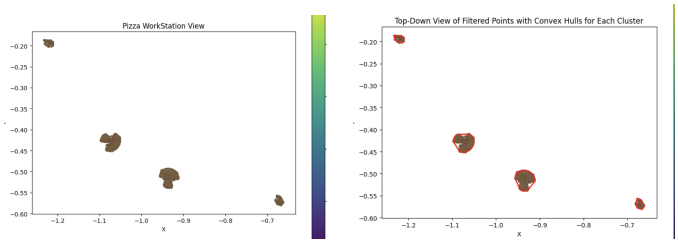
To set up this area we use a python script that generates a YAML file for our simulation. This enables us to change some

key variables across the model directives, like the number of tomatoes and mushrooms, placement of bowls, etc. During development, the YAML generation script also enabled us to invoke our set up with or without all objects to promote faster iterative testing; once a desired trajectory was evaluated without objects, objects could be added to the scene. The YAML file also contains plant configuration details. While we initially invoked simulation with hydroelastic contact, long simulation times without measurable performance increases in our environment drove a late decision to return to point contact force simulation.

C. Perception

Our perception pipeline is designed to capture the "pizza state," including the number of ingredients on the pizza and the overall pizza coverage they provide. The RGBD camera captures a depth image of the scene and generates a point cloud of the pizza pan. The point cloud data is then used to assess the ratio of ingredients per pizza area. We use RGB data to estimate the ratio of ingredients to overall pizza area. The tomato, mushroom, and dough all have a different color signature, so we can use a "color ratio" of each ingredient's color to the background dough color.

- CameraSystem Class: we use this class to gather most of the key data from the camera. This includes accessing and transforming the depth image from the camera, an RGB image from the camera, and a corresponding point cloud. The class also visualizes the point cloud on the scene. This class directly links to our system's output ports.
- Filter point cloud by color: we mask points of the point cloud using specific RGB data. Each point of the point cloud contains registered RGB data, so only points matching the target color will be used to calculate our desired ingredient coverage ratio.
- Pizza area calculation: we use a circle-fitting function applied to a point cloud filtered on the white dough color, and we use that information to get the area of the dough. While we currently use the same dough object in each simulation, our approach generalizes to account for slight variations in pizza size within the camera's field-of-view. In this way, the ingredient coverage ratio is tailored to the size of each pizza individually. This approach also generalizes to support various pizza sizes.
- Ingredients clustering and fitting algorithms: Once we isolate the desired color, we attempt to form clusters representing individual ingredients. We use a Density-Based Spatial Clustering of Applications with Noise (DBSCAN) algorithm to form individual clusters. We selected this algorithm because of its noise robustness, ability to operate without a known number of initial clusters, and existing Python implementation. [5]. Next, we then use Andrew's monotone chain algorithm to find the 2D convex hull of each cluster. An example of this convexification applied in our environment can be found in Figure 2b. A convex hull is the smallest convex boundary that encloses a set of points to form polygon. We selected this method for its deterministic behavior, efficiency, and existing Python implementation examples [4]. Once the convex hull is formed, we find the estimated ingredient total area as the sum of polygons.
- Final Calculation: To calculate the overall ingredient coverage, we define a minimum ratio of ingredient area to pizza dough area: in our environment, for example, we define that at least 13% of the pizza shall be covered in mushrooms. This approach allows us to assess ingredient coverage based on the "amount of ingredients" rather than the number of ingredients. In real-world applications, many ingredients vary in size and shape, and our method ensures that each pizza really is sufficiently covered. One drawback to this approach is the algorithm's difficulty differentiating ingredients of similar colors; if multiple ingredients share similar colors (e.g. green peppers and spinach), the algorithm will have trouble discerning the two classes of objects. Deep perception and other image segmentation methods may more successfully identify multiple object classes.



(a) Clustered Mushrooms. Results of point cloud filtering by color to cluster pixels likely attributable to mushrooms.

(b) Convex Mushrooms. Outcomes of mushrooms as convex hulls.

Fig. 2: Side-by-side comparison of Clustered Mushrooms and Convex Mushrooms from the perception pipeline.

D. State Machine

Our high-level planner follows Finite State Machine (FSM) logic implemented in the PizzaPlanner class to calculate trajectories, estimate ingredient coverage and overall pizza area, and switch control modes. As inputs, the PizzaPlanner accepts robot state and body poses from the plant, rgb and depth images from the camera, and a point cloud from the point cloud processing system. Outputs from the planner comprise joint-space desired positions (for global IK control), desired end effector pose and parameter controls for the DiffIK controller, and reset/port control logic for swapping between DiffIK and IK controllers. The State Machine contains initialization logic, port update helpers, function wrappers for port manipulation, and main loop code. Key challenges of our implementation included the development and connection of a custom LeafSystem from scratch, port updates for stored state variables, piecewise trajectory execution from within a single state, and handling of multiple gripper actions and iiwa arm

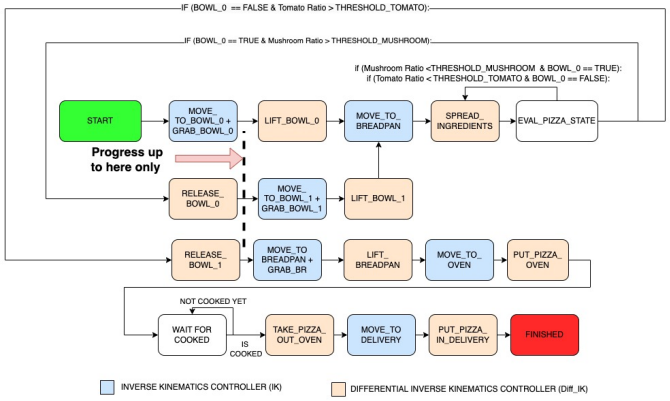


Fig. 3: Current State Machine. The dashed bold line shows the current state of our PizzaPlanner system. With our current architecture, the remaining states are defined in the planner but remain untested at report time.

actions within a single state. The planner we present contains the state logic shown in Figure 3; while only a subset of these states are implemented and tested at this time, the planner's architecture is quite extensible to an unlimited number of states, state logic, and transitions between differential IK and full IK operations.

E. Trajectories

The PizzaPlanner System smoothly transitions between end-effector trajectories generated for differential IK control and joint-space trajectories generated for global inverse kinematics control. Additionally, we implemented functions that allow the planner to dynamically modify the DiffIK controller's parameters for joint position, joint velocity, and end effector velocity limits on the fly. Specifically, this tool helps us lock the mobile iiwa base to a single position or along a certain degree of freedom. To generate end-effector trajectories, the Planner computes a linear series of timestamped PiecewisePoses. At each timestep, the Planner publishes the next desired end-effector pose to its desired_pose port. The DiffIKIntegrator accepts this pose and executes the incremental move.

One unique challenge of our project was creating an "ingredient spreading" trajectory which moves the end effector along a sinusoidal path over the pizza. The trajectory generator accepts several arguments, including amplitude, frequency, and duration (number of keyframes). Similar to the the IIWA_Painter example, we interpolate the trajectory. A differential IK trajectory executor accepts the arbitrary trajectory of end effector poses; in a similar way to the position-controlled moves described above, each desired pose is passed to the robot in time. The sinusoidal trajectory effectively shakes the bowl to spread ingredients, but the conclusions may not be applicable to a real hardware iiwa system; to ensure that ingredients were spread effectively and the gripper maintained control of the bowl, we heavily modified the coefficients of static and dynamic friction of both the mushrooms and each bowl for optimal pickup and spread characteristics.

Larger moves between work areas are implemented by the inverse kinematics solver in the PizzaPlanner module. Like the differential IK operation mode, the global IK solver has been modified to accept base fixture conditions for the XYZ translation axes. These are implemented as additional equality constraints solved during simulation. To ensure sufficient clearance from the mobile base to the workstation tables, we also implemented a bounding box constraint applied to the XY translation of the robot keeping the mobile base within a central work area. With these constraints, the IK solver returns desirable poses. In most positions, the robot maintains good range of motion over a selected work area and remains in a position desirable for the next move within the environment. After transitioning to inverse kinematics control, the Planner begins to publish the full IK optimization solution as a PiecewisePolynomial 10-dimensional trajectory (3 translational "joints" plus 7 iiwa joint angles). The iiwa plant's InverseDynamics controller actuates the desired joint angles for a desired robot state (via the StateInterpolator described in subsection III-A).

RESULTS

We focused heavily on environment setup, execution of mobile_iiwa actions, pizza ingredient coverage perception, and state machine logic in this project. The varied technical challenges of each of these elements limited our final solution to a minimum viable ingredient-spreading and perception routine. However, our perception and ingredient-spreading loop demonstrates the effectiveness of an automated food assembly station. In 28 seconds, we can effectively cover 14% of the area of the pizza with mushrooms using two passes: the first pass covers 12% of the pizza area, while the second pass completes our requirement. This is roughly 7 mushrooms. For manual pizza assembly, we can estimate placement of each mushroom will take at least 6 seconds, for a final ingredient retrieval and placement time of 42 seconds. Placing ingredients closer to the pizza would only improve the efficiency of operations. Our entire simulation time has been also optimized to run in around 10 minutes with over 70 items (plus the mobile_iiwa itself) in the scene. Our perception path accurately estimates the amount of ingredients on the pizza, showing viability of our ingredient identification method.

Completion of our ambitious initial goals and timeline were dramatically impaired by the complexity of our implementation. We underestimated the complexity of the hardware/diagram setup, state machine development, the time required to effectively ramp up and utilize the robot, and the difficulties of grasping and manipulating objects under realistic physics conditions.

NEXT STEPS / CONCLUSION

While our system serves as a good proof of concept, it requires many next steps, including completion of our existing plans and the refactoring of several code elements.

First, we would like to add tomatoes, put the pizza in the oven, and deliver the cooked food to the delivery area. This

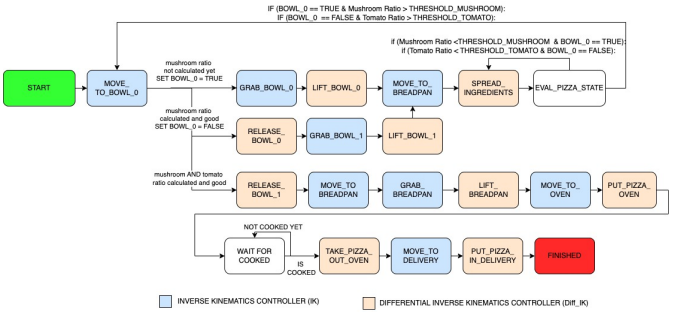


Fig. 4: Future State FSM. Improves transition state logic to apply more ingredients to the pizza while splitting out gripper and arm repositioning actions to separate states.

would complete our intended state machine work expressed in Figure 3 and provide a "full" solution for a singular pizza making cycle. We can also improve on this project by adding sauce and a third ingredient (like cheese) to make the complete pizza a tired graduate student will truly enjoy. Extending our methods to coordinate the simultaneous production of several pizzas may produce true time savings at scale. Improvements to our perception approach may add a camera on top of the ingredients table to monitor ingredient levels and use that feedback to increase or decrease the amount of bowl shaking required. Training a deep perception pipeline may also allow us to estimate coverage of simultaneous ingredients to save time in the assembly process.

Our code base and state machine require code cleanup, README documentation, and general state logic merging and consolidation. Our future finite state machine Figure 4 represents an achievable future state for the project.

To conclude, this project is a proof of concept demonstrating the ability to assemble a pizza more efficiently. However, the technical complexity of our delivered solution does not match the complexity of the actions it performs. There exist several ways to demonstrate similar performance without the implementation time we applied. While a state machine and fully implemented LeafSystem make our solution far more extensible and complete, many of the same actions we perform could have been achieved with direct access to diagram context and executed through Jupyter notebooks (as shown in course materials). Robotic manipulation advances by simplifying movements and using the environment to our advantage. We hope that this project will demonstrate the complexity of a robust implementation to solve even simple robotics problems and the benefits that may come with ever-advancing frameworks and software tools to democratize the development of robotic systems.

Team Member Contributions

Marine and Matt contributed equally to this project, dividing responsibilities up by subject area and tailored to each team member's skill set. Marine owned perception and environment setup, while Matt owned state machine set up, diagram connections/port mapping, and trajectory enablement. Both collaborated constantly, writing a lot of the coding together and broadly sharing learnings across our team. We worked in both a local VSCode environment and a Deepnote cloud notebook server. Github Copilot Chat was used for debugging throughout this project, but solutions and models are our own or by attribution.

Acknowledgment

We thank Professor Tomas Lozano-Perez and the entire Robotic Manipulation teaching team for their help throughout this course and project. Finally, we issue a deeply heartfelt thank you to *Nicholas Pfaff* for his endless support, timely responses, and unwavering commitment to helping us through our many challenges.

References

[1] Russ Tedrake, "Mobile Manipulation", Chapter 7 in *MIT Manipulation*, https://manipulation.csail.mit.edu/mobile.html
[2] Deepnote, "Mobile Base IK Example", https://deepnote.com/workspace/Manipulation-ac8201a1-470a-4c77-afd0-2cc45bc229ff/project/e969edb2-a7e6-480b-b81d-4d5778c62ec9/notebook/mobile_base_ik-f6c05c387a314333961ca60aeed3c84a
[3] Peter Holderrieth, "Robot Librarian", https://github.com/PeterHolderrieth/RobotLibrarian/tree/main
[4] Wikibooks, "Algorithm Implementation/Geometry/Convex hull/Monotone chain", https://en.wikibooks.org/wiki/Algorithm_Implementation/Geometry/Convex_hull/Monotone_chain
[5] Wikipedia, "DBSCAN", https://en.wikipedia.org/wiki/DBSCAN
[6] Russ Tedrake, "Pose Estimation", in *MIT Manipulation*, https://manipulation.csail.mit.edu/pose.html
[7] Deepnote, "Segmentation and Grasping Example", https://deepnote.com/workspace/Manipulation-ac8201a1-470a-4c77-afd0-2cc45bc229ff/project/c91043b1-e029-4b16-9924-9f7aca75a010/
[8] Russ Tedrake, "Segmentation", in *MIT Manipulation*, https://manipulation.csail.mit.edu/segmentation.html
[9] Russ Tedrake, "Clutter Clearing", in *MIT Manipulation*, https://manipulation.csail.mit.edu/clutter.html
[10] Nepfaff, "Open Loop Planar Pushing", https://github.com/nepfaff/iiwa_setup/blob/main/iiwa_setup/controllers/open_loop_plannar_pushing.py
[11] Russ Tedrake, "Clutter Clearing Notebook", https://github.com/RussTedrake/manipulation/blob/master/book/clutter/clutter_clearing.ipynb
[12] M. M. Mepani, K. B. Gala, T. A. Mishra, K. Suresh Bhole, J. Gholave, and S. Daingade, "Design of robot arm for domestic culinary assistance," Materials Today: Proceedings, vol. 68, pp. 1930–1945, 2022, doi: 10.1016/j.matpr.2022.08.140.
[13] "SpaceX rocket scientists built a robot that makes $8 pizzas - Los Angeles Times." Accessed: Oct. 14, 2024. [Online]. Available: https://www.latimes.com/business/story/2022-06-16/stellar-pizza-robot-story
[14] Shreya Chaudhary (MIT), [6.421] OrderBot, (Dec. 12, 2023). Accessed: Oct. 14, 2024. [Online Video]. Available: https://www.youtube.com/watch?v=zOvT7VnuZ2k
[15] Drake Development Team. *Drake: Model-based Design and Optimization for Robotics and Control*. Massachusetts Institute of Technology, 2024. Available at: https://drake.mit.edu. Accessed: 2024-12-09.