Toward Scooping In Robotics

Martin Chan
MIT EECS
Cambridge, MA
martinch@mit.edu

Fiona Gillespie
MIT EECS
Cambridge, MA
fgillesp@mit.edu

Hannah Kim
MIT Economics
Cambridge, MA
hann@mit.edu

***Abstract*—In this paper, we investigate the ability of a fixed robot arm to reliably scoop granular objects from one selected bin to another. Scooping has the potential to increase the range of object sizes and shapes that a robot arm can interact with, but it is a largely unexplored application in robotics. We build on existing work in grasping and pick-and-place tasks, focusing instead on a scooping motion to pick up multiple smaller granular objects at once. We develop a full-stack system for an iiwa robot arm that can be directed to scoop granular spherical objects from one bin to another using a scoop that is welded directly to the end of the arm. This work primarily focuses on using pre-planned trajectories, but in the future, live feedback could be incorporated through vision systems or sensor suites. Our system is able to reliably scoop 3 spheres from the first to the second bin, using both a geometric and teleop playback approach to create pre-planned trajectories.**

I. Introduction

Going into this project, we wanted to explore everyday applications of robotic manipulation, such as in a household kitchen. We chose scooping, or the act of picking up and moving something with a scoop, as an interesting, common action that seemed to be largely unexplored in robotics academia, as opposed to pick-and-place actions with a gripper. Our initial goal was to have an environment setup where a robot could be directed to scoop a given volume of a granular good (such as beans, rice, and maybe flour) to move it from one bucket to another using a measuring cup, ladle, or spoon.

We expected to have to re-scope as we discovered the limits of our own (and Drake's) abilities as beginners to the tech stack. We did this very early on, as our initial project proposal explored the mechanics of chopping carrots. After receiving technical feedback on the scope of this project, we reoriented ourselves to scooping so as to avoid having to deal with fracture mechanics and otherwise dynamically recreating meshes. In particular, we received a suggestion to try and replicate an existing strategy of pre-computing trajectories and playing them back according to the state of the bin being scooped from. Although we didn't manage to automate our trajectory selection, we set up

The main technical problems we expected to encounter with scooping were in the modeling of contact forces between the objects being scooped and the scoop itself, as well as creating the scooping trajectories the robot arm would be following.

6.4210 Robotic Manipulation

II. Prior Work

There is a growing field of robots in kitchen applications, such as with Dexai Robotics's Alfred (dexai.com/meet-alfred), which has scooping capabilities. The Toyota Research Institute (TRI) has also done work on food preparation, including simulations with hydroelastic modeling of spatulas scooping up carrot-like objects (and other shapes) off a table. Spyce, a Boston-based startup by MIT graduates, operated a restaurant in Boston with a robotic kitchen. They've since been acquired by sweetgreen.

To the best of our knowledge, there is no existing academic publication on scooping with a concave tool by robots. The closest we could find was Scooping Manipulation Via Motion Control With a Two-Fingered Gripper and Its Application to Bin Picking, a paper published July 2021 [1] which examined the act of scooping a small flat object (such as a coin) off a table with a thin flat spatula.

We were told by Russ Tedrake that some prior approaches to scooping involved pre-computing appropriate trajectories for different bin states (such as empty, half-full, full) then using perception to determine which trajectory to select. This was the direction that we hoped to bring our project, though we did not get far enough for the perception selection process.

III. Approach

A. Overview

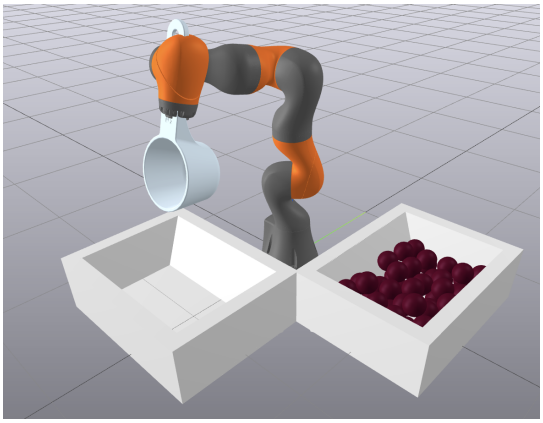

Fig. 1. Environment setup: iiwa arm, scoop, bins, and objects being scooped.

Our environment setup, shown in Fig. 1, consists of the robot arm doing the scooping, the objects being scooped, and

the bins holding the scooped objects. Having worked almost exclusively with the iiwa arm for class assignments, we chose to use the Kuka LBR iiwa robot arm, a 7-axis robot arm with joint-torque sensors in all axes and relatively robust support in Drake.

We decided to try an approach suggested to us in the first project check-in where we pre-compute trajectories and choose them to execute. Although we wanted to try to get a perception module to choose from trajectories automatically based on the state of the bin, we only had enough time to save and execute trajectories manually.

We used two parallel approaches to create our trajectories. One approach was geometric using an approach similar to the robot painter from the class homework, where we first encoded key poses from a semi-circle then interpolated them, adjusting parameters (like radius) manually based on what we saw in the simulator [2]. The second approach was to teleop-record, marking down key poses along the way, then exporting those key poses for later reuse. We improved our teleop-recording process from simply printing the pose every couple seconds, to printing the pose on command [, to registering poses then exporting the corresponding Python code in text, to exporting the poses as code.]. Because we knew our work would be iterative, we tried to place special focus on tools that would help us in development.

One of our issues with our project was the slowdown from our need to simulate many bodies (upwards of 150 spheres) in order to have a convincing amount of substance to scoop. To that end, we tried two approaches: one, we changed our contact solver from Tamsi to Sap (upon Russ's recommendation), and we also reserved our full-bin simulation for integration tests only. While we would've liked to use a full-bin for testing our initial trajectories, it would take impractically long to simulate the full set of bodies as part of our main iterative process. Our pre-planned trajectories still performed well on the full-bins during testing, despite being planned with fewer spheres.

B. Environment Setup

To create the environment shown in Figure 1, we needed to choose a model to be our scooper. We decided to use a measuring cup as the scoop so that it would have edges (we were a bit worried about the objects falling off the side of the spatula) [3]. We used the mesh-to-sdf-converter from the Authoring Multi-Body Plant tutorial to create the collision geometry for our measuring cup [4] [5]. As we progressed in the project, we also experimented with the right scoop size for our set-up. Our initial scoop was too large to properly fit in the bin, but we wanted a large enough scoop that we could have

Welding an imported scoop to the end of our iiwa arm avoided the added complexity of directing the iiwa arm to grip a scoop. Both this and our bins of spheres were added through model directives.

Our first trials used small rectangular bricks as the objects to be scooped, but we eventually transitioned into using spheres due to simpler collision geometry, which significantly improved the simulation speed with a visually comparable number of objects. Combined with a faster contact solver and a meshcat update that got pushed out during our development process, we saw huge improvements in our simulation runtime. Still, there's a tension between having few enough objects in the environment to quickly simulate, and enough objects so that the simulation can both reliably scoop and look impressive while doing it. To make testing feasible, especially for initially saving our trajectories, we teleoperated in an environment with either no spheres or as few spheres as possible.

We can still use these trajectories in simulation with many objects as long as there are enough objects to act roughly like a fluid. If the scoop could only fit a single ball, then we can't guarantee that a saved trajectory that was recorded successfully scooping a sphere would do it again with a different random configuration of spheres. Most of our trials use 50 or fewer objects, since it is about $O(n^2)$ time to simulate n objects.

One of the most important parts of environment setup is actually placing the objects into the world. All objects "spawn" at the origin at the start of the simulation, and move into their starting poses as the simulation proceeds. Poses and welds for our bins, iiwa arm, and scoop are all defined in the model directives for the environment. For the spheres, their initial position is suspended above the bins (shown in Fig. 2), and then they fall into the bin as the simulation proceeds as seen in Fig. 1.

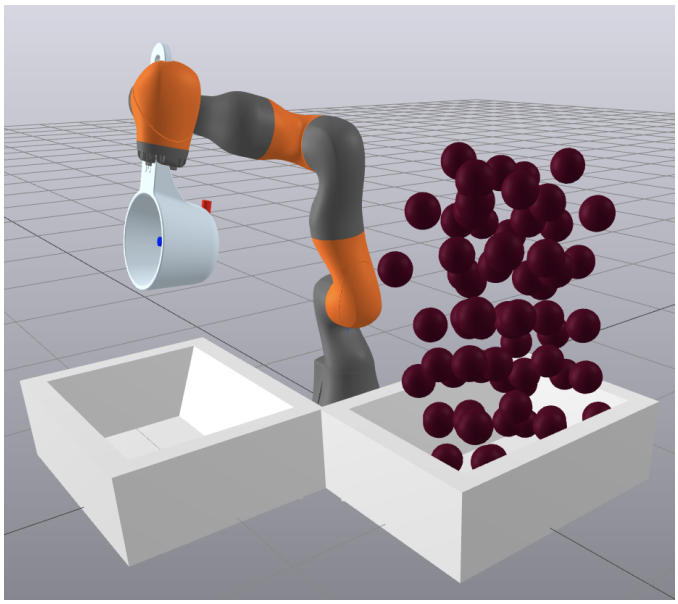

Fig. 2. Environment before simulation begins advancing and balls fall.

C. Spatial Context

Before going deeper into our approach, we would first like to cover some relevant context about how to think about trajectory planning with geometry.

Our iiwa has a lot of capability with its 7 degrees of freedom. If we know where we want the scoop to be, how do we tell the iiwa to do that? This is where *frames* become

useful. Frames are a way to abstract out some of the geometric complexity, and just focus on the location of some important elements in our system. We can define the *world frame* as the "center" of our system and the origin for all other definitions. It is also very useful to have a *scoop frame*, which lets us define a new coordinate frame located at the cup of the scoop. Then, instead of having to worry where the scoop is with respect to the origin, we can just decide what location and orientation we want the scoop to be, in the scoop coordinate system. The scoop frame's origin can be defined by a translation and rotation with respect to the world frame. Fig. 3 shows us both the scoop frame, located in the cup of the scoop, and the world frame, extending out of the iiwa arm base. Having a scoop frame is especially helpful for geometric reasoning about which direction we want the handle of the scoop, or which way the open face of the scoop should point.

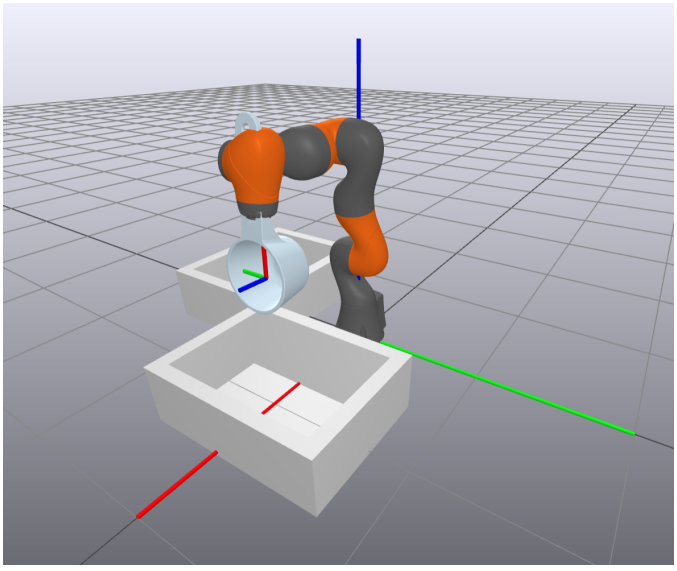

Fig. 3. The scoop frame can be seen inside the cup. The x-axis (red) is aligned with the scoop's handle, the y-axis (green) is aligned with the right side of the scoop, and the z-axis (blue) is aligned with the open face of the scoop. The world frame is shown at the base of the robot

Another important geometric concept is a *pose*, which just describes a position and rotation of an object. When we have a desired scoop pose in the scoop frame, we can use the pose of the scoop frame's origin to convert back into world frame as well. To learn more about frames, poses, and transforms, see the corresponding sections from the Basic Pick and Place chapter of the Robotic Manipulation textbook [2]. Now that we have covered some background, we will discuss our two trajectory planning approaches in depth in the following sections. We developed both of these approaches in parallel to scoop from one bin and pour the spheres into the other bin.

D. Geometric Trajectory Approach

Our first approach investigates planning a scoop and pour trajectory by estimating each motion as part of a simple geometric shape.

1) Geometric Scooping: Scooping roughly looks like moving in a semicircle, so we used this as our starting point. Using the scoop frame, our first goal was to create a semicircular sequence of *key frame poses* we would like the scoop to move through. Instead of defining every single pose that the scoop is in, we can actually generate just a few poses, and use *interpolation* to fill in the rest. The basic idea behind interpolation is connecting the dots between our key frame poses. Since we are moving in a circle in 3D space, we use *spherical interpolation* to best represent what happens between our key frame poses. Fig. 4 shows a sequence of circular key frame poses. There are 10 to make it extremely clear what the path would be, but the trajectory could still work well with a smaller number of frames.

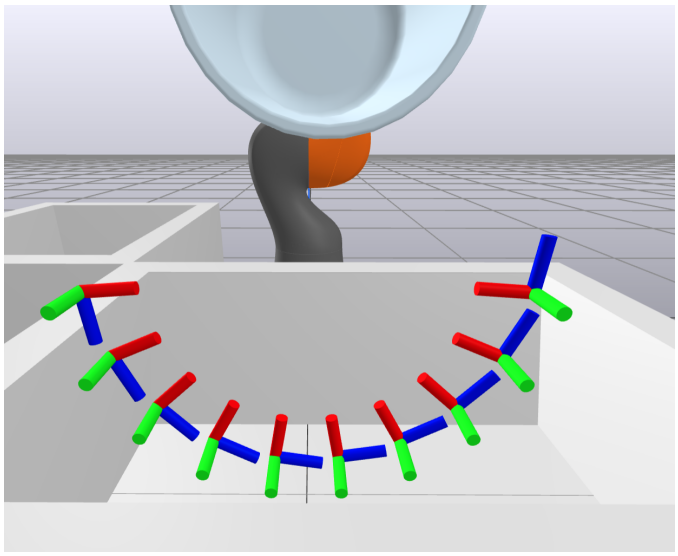

Fig. 4. A circular path of 10 key frames

In order to create this circular trajectory, we referenced the robot painter notebook from the Basic Pick and Place chapter of the course textbook [2]. As one of our assignments during the course, we created a function to compose circular key poses (given a list of angles, the pose of the center of the circle, radius of the circle, and where the arm starts from). This function creates a transformation for each of the angles: the rotation is determined by the angle, and the position moves away from the center in the rotated radial direction. This transformation can then by applied to the center of the circle to create a new key pose on the circle. To verify that the trajectory is as intended, it is very useful to visualize the key pose. This visualization is how the key poses are displayed in Fig. 4.

Once we have the key frame poses, the final step is commanding the scoop to be in these poses. The process of going from poses to joint angles is known as *inverse kinematics*. Since we would like to move to a pose, we need to learn how *changes* in a pose affect the joint angles. This makes the process rely on derivatives, or *differential inverse kinematics* [2]. We used a differential inverse kinematics solver frame

class. We passed this solver the scoop frame as its reference frame, and then it controls the iiwa to manipulate joints such that the scoop lines up with the commanded trajectory. After this, all that's left is tuning the trajectory!

As a brief recap, here are the 5 main steps to create the geometric scoop trajectory:

1) define scoop frame
2) create circular key frame poses
3) visualize key frames
4) solve for joint angles with differential inverse kinematics
5) tune trajectory

Once the basics of this geometric approach were implemented, the next step was reviewing the scoop performance and making adjustments as needed. The circular trajectory from Fig. 4 made a decent attempt of scooping the spheres, but sometimes struggled with being able to interact and pick up the spheres (they can get pushed away by the scoop).

For extra flexibility, we created an elliptical key frames function. Instead of a radius, this function requires the lengths of the semimajor axis and semiminor axis. Now, we had much more control over our trajectory, and we could get the scoop closer to the bottom of the bin (shown in Fig. 5). Note how different the edges of the elliptical path key frames in Fig. 5 are from the circular path key frames in Fig. 4.

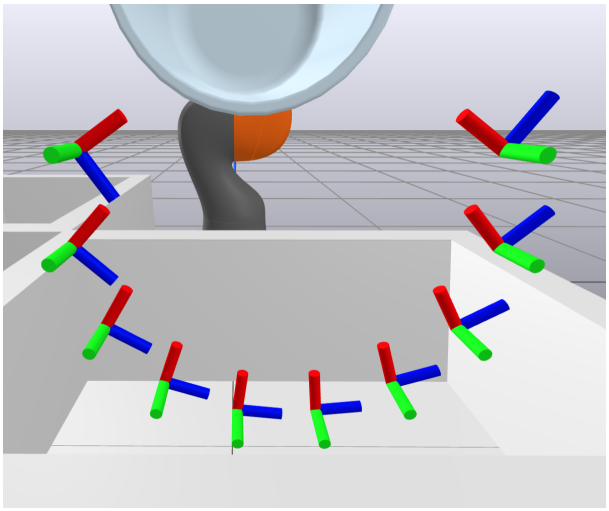

Fig. 5. An elliptical path of 10 key frames

2) Geometric Pouring: After we created a scoop trajectory, the next step was creating a trajectory to pour the spheres into the other bin. Once the spheres are already in the scoop, we have to be more careful about how we command the scoop so we don't drop our spheres. In addition, there were some limitations in terms of which poses the arm could reach as well. Sometimes it seemed like two key frames would be easy to go between, but the solver would get stuck. After trial and error, we created the trajectory shown by key frames in Fig. 6.

After finishing the first scoop, we added a key frame to ensure the z-axis of the scoop stays upwards, as shown in Fig. 7. This helps the scoop avoid tipping and dropping the spheres).

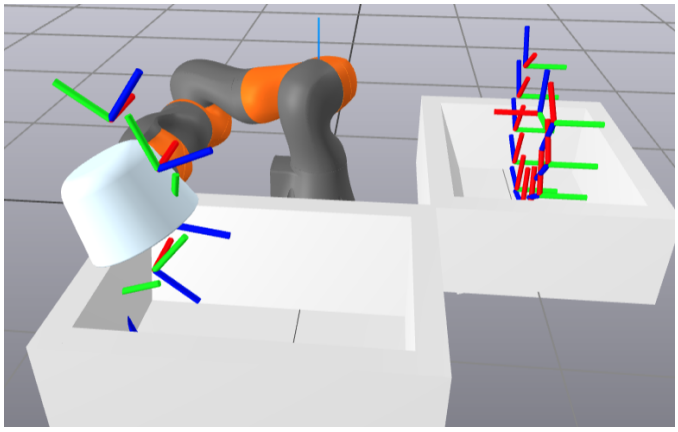

Fig. 6. The entire scooping and pouring process visualization. Right: scoop path. Left: pour path

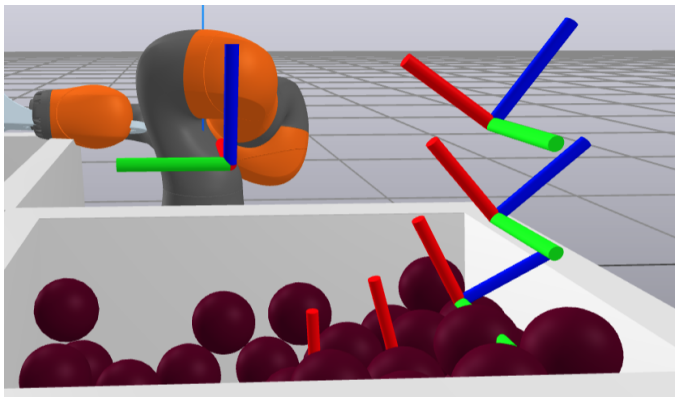

Fig. 7. The key frame on the left keeps our scoop upright so that we don't drop any of our spheres

This current visualization of key frames we use for the pouring motion is displayed in Fig. 8. It was easiest for the iiwa to pour from the left side of the bin. When the key frames commanded the right side of the bin, the arm moved around unhappily.

E. Teleop Record Details

Early in the project, we determined that a solid starting point for our system would be to use pre-computed trajectories. While our ideal system would either not be teleoperated at all (if we managed to get to a perception system) or teleoperated only on a very high level (by trajectory, not by pose or joint commands), we wanted to explore using teleop as a way of creating trajectories. For that, we needed a way to save and playback teleop trajectories across simulation runs.

This approach gave us two things. First, playback removes the need for the robot to compute a new trajectory with every scoop. Second, it also allows us to test quicker since the addition of the scooping objects dramatically increases the simulation time.

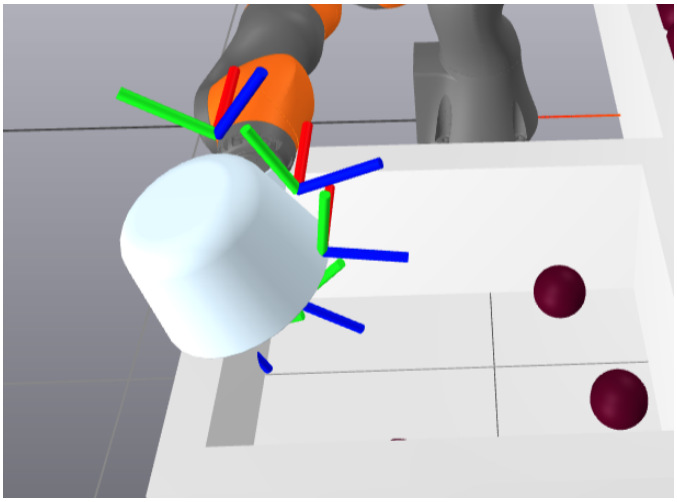

Fig. 8. The scoop pours the spheres into the bin from the left side. This trajectory was created with trial and error

We began by setting up our teleop environment to use pose sliders on the last link of the iiwa. We could have instead used pose on the scoop, but we decided that using this pose was both easier to implement and allowed for a more human-like scooping motion (intuitively, we scoop centered from our wrist, not the head of the scoop!) We could have also instead used joint sliders, but teleop comes more naturally to us with poses.

Once we got the teleop to successfully scoop spheres manually, we began considering how we could build our trajectory from teleop. Our naive approach was to first print out poses (e.g. `RigidTransform(RotationMatrix(0, 0, 0), [0, 0, 0])`) in the code output box whenever we pressed a "Print Pose" button. The idea was to put all these printed poses into a code block that we would execute to create a trajectory object. This seemed inefficient, so we then investigated how to save all the poses for a given trajectory through files.

We later created a `SlidersMemory` Python class that would maintain the history of poses for the robot during a simulation and respond to a "Save Poses" button that would write them to file. Our file representation was a text file with a 6-tuple on each line representing each pose's 6 parameters (3 for rotation, 3 for position). The idea was that a trajectory should be reproducible by saving the poses from a teleop in a particular way and playing them.

We needed to decide which poses to save. Initially, we thought to have each pose differ from adjacent poses by only one value, representing a single slider changing. The idea was that we would operate only one slider at once, and that we would be able to play back faster than we made the trajectory, since we might slowly change one slider during sensitive parts of the movement like aiming the scoop.

Further testing showed that we often teleop using two sliders at once, like a rotation and a position in the case of the actual scooping and dropping process, so we needed to change our pose saving strategy. We switched to saving the pose by time step instead, but we still needed to manually clean up our trajectory files to make them smoother played back. We suspect that some further work on which poses to save and recovering trajectories from saved poses can make the replay more reflective of the original teleop.

The main components to using the teleop record for future automatic playback are in cleaning up the teleop path, reading the files they're stored in, and actually moving the iiwa arm based on the information read. A single trajectory file will usually hold an ordered list of desired poses for the iiwa. Often there are more than a few unnecessary movements in a single file, and so (barring more robust write conditions) some manual cleaning is desired to create smoother trajectories. In addition, deleting and adjusting poses can help with resolving issues from the iiwa getting "stuck" or having joints that are fully extended. The files are parsed to create paths based on movements between poses, rather than the positions themselves. The iiwa can then be directed to execute these movements, whether it is just once or on loop.

This execution is not necessarily teleop at all, but simply a pre-computed path that has been passed into the iiwa arm. In some cases it was computed from a teleop trajectory, in other cases just manually created, and could even come from the geometric scooping approach.

Being able to play any given path yielded some key lessons for our trajectory planning and environment simulation. It matters a lot what the previous position of the robot is; the robot put in the same location and rotation positions look very different coming from two different previous positions. We found significant value in having moves along multiple axes at the same time, especially as the actual "scooping" motion is deceptively complex.

F. Challenges and Learning Experiences

Some of the toughest parts of the project weren't directly related to robots, but had more to do with the development process in general.

We partially set up a version control system through Git, but did not use it or branches as liberally as we should have. Instead, our best version control was an ad hoc system of copying files and using Deepnote's native autosave and revert tools. This proved to be unfortunate during the debugging process and also led to us having way more notebooks than we were using at a single time.

Some debugging was tough. We made a change to our scoop sdf later in testing that broke our universal environment. It took an hour for us to figure out that it had come from an effort to make the scoop lighter that inadvertently had made it heavier. To try and fix it, we reverted our notebooks to the last known working version, but Deepnote's history only tracks the notebooks, which the scoop sdf file was not part of.

Since our project requires importing a scooper object, we were surprised by the difficulty of setting up the `package.xml` and other boilerplate necessary to import it

into the simulation. It ended up being simple to put in, but we couldn't figure it out on our own before asking for help.

This project was also our first time really working through understanding all the code that goes into running a Drake simulation. We had an early hurdle figuring out how to adjust our code to accommodate the `MakeManipulationStation` versus `MultibodyPlant`, though it was a major help having sample code from the textbook for some parts. It is a very different experience just reading the code that sets up the psets, compared to actually having setting up our own simulation with custom files. Another hurdle was knowing how to modify the controller plant since we were not using the gripper (big thank you to Russ for the individual help on that!). In hindsight, the setup does make a lot of sense, but it was a process to really understand what was going on the first time through.

For some parts, the documentation wasn't much help, but we often looked through some Drake source code on classes like `MakeManipulationStation` on Github to try and see how we could implement the things we wanted to implement.

It became quite time-consuming to work in the simulation environment once there were a large number of objects within the environment. Even after updating to use the SAP contact solver, which is faster than the default meshcat contact solver, we found that increasing the number of objects getting scooped slowed down the simulation significantly, such that it could take nearly 10 minutes with 50 spheres to simulate just one or two scoops. This became a pretty strong constraint for how we thought about the granularity of poses in a single trajectory–too few poses, and the arm would not be able to fully execute a scoop; too many poses, and the simulation would take unreasonably long.

IV. Results and Discussion

A. Geometric approach

Preliminary results of the elliptical scoop and quarter circular pour were very promising. Trajectories were planned first without spheres and then tested with approximately 50 spheres. The quantity of spheres significantly slows down the runtime of the simulation. This trajectory reliably picked up and poured 3 spheres from the bin of 50. Fig. 9 shows the environment after a successful scoop and pour.

While picking up the 3 spheres, several spheres were also pushed out of the bin and into the world. When the same trajectory was tested with only 30 spheres, it did not successfully pick any up. This trajectory needed enough spheres in the bin to be able to scoop some off the time - if there were only 30 spheres they could easily be pushed out of the way by the scoop. It is likely that this trajectory would work with more than 50 spheres, but would not generalize well to an emptier bin. We did not do much testing on successive scoops, but it would likely work if the bin is still full enough. As the current trajectory does lose some spheres to the floor on every scoop, it would be better to fine-tune this trajectory before performing subsequent scoops.

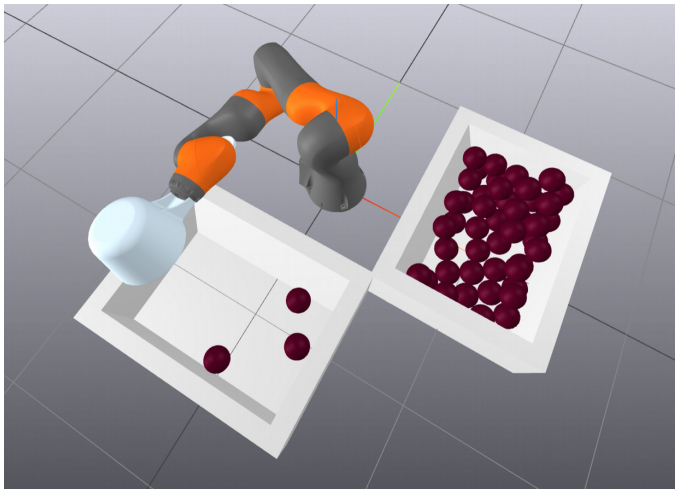

Fig. 9. The geometric approach reliably scoops and pours three spheres from one bin to another, when starting with 50 spheres.

The main hurdle in setting up the geometric trajectory was initially performing inverse kinematics on the last link of the iiwa. However, this was unintuitive for creating key frames. Once we correctly set up a frame for the cup of the scoop within the scoop sdf and ran inverse kinematics on the cup, the geometric planning worked much better (big thank you again to Russ for advice on this!).

It was an interesting learning experience trying to set up the pour following the scoop. The first attempt had the scoop move to the corner between the bins, which is on the right side of the bin to pour into. However, the iiwa kept getting stuck and could not pour from this side. So, the next attempt had the scoop move around from the right bin counterclockwise to the left side of the left bin. It seemed like it would be helpful to put the scoop close to where the pour trajectory would start. However, the iiwa would get stuck once again. It turned out that the iiwa was actually happiest letting the interpolation handle the movement between finishing a scoop and moving to pour the spheres into the next bin. The attempted intermediate frames made it harder for the iiwa to perform the pouring. This was an interesting result that we were not expecting when creating the path. Perhaps it makes sense to require as little as possible, and let the iiwa have freedom to move whenever it can.

B. Teleop Playback

The process of generating our scooping trajectories was fairly user-friendly. Our teleop setup is shown in Fig. 10

We used the keyboard for teleop, which works smoothly with only a little bit of practice. To save, we just press a button and the trajectory is saved to a file. The goal was to make creating a trajectory as easy as doing teleop for that trajectory, which we made mostly true with our tooling, though some manual cleaning is necessary to smooth out some trajectories. We expect some further work on the trajectory file representation on both saving and playback to remove the need

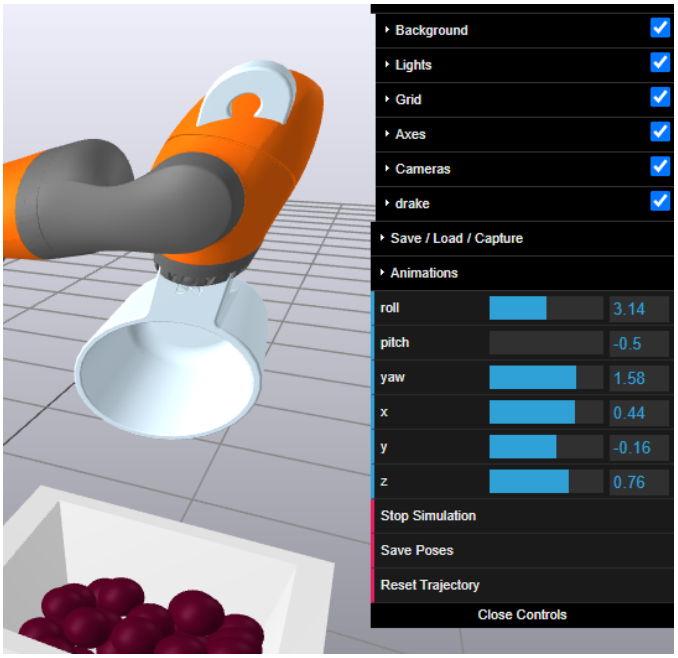

Fig. 10. Our teleop allows the user to control the pose of the scoop via joint sliders. There are also buttons to save poses and reset the trajectory.

for manual cleaning and to make creating reusable trajectories even more user-friendly.

We got strong results playing back precomputed trajectories. Like with the geometric approach, trajectories were tested first without spheres and then with approximately 50 spheres. This trajectory reliably picked up 2-3 spheres (depending on how they had landed) from a bin of 50, and poured them into the neighboring bin. The iiwa would usually push a couple spheres were also pushed out of the bin and into the world. After the first scoop, the iiwa can usually get 1-2 spheres, but struggles more with this as the bin depletes.

C. Discussion of both approaches

While the geometric approach creates relatively smooth trajectories, navigating the bin environment proved difficult. The teleop approach creates much more flexibility and can quickly plan trajectories to new situations. The geometric approach instead requires careful planning ahead of time, but creates potentially smoother or more elegant trajectories, especially if an optimal ”shape” of the trajectory is known.

We expect that the scooping performance should improve as objects get smaller in relation to the scoop. When there are many small spheres rather than fewer large spheres, the scooping action relies less upon the specific configuration of spheres in the bin. Asymptotically, we expect our system to work better as the scoopable objects in the bin approach a fluid. However, more objects (and contact surfaces) require more time to simulate.

The main bottleneck in our simulation environment is having to simulate all the little objects in the bins. If we had a physical robot, we could more easily perform tests of our precomputed trajectories with full bins. However, even outside of testing with full bins, we had reason to believe that the robot should behave similarly whether it was moving through air with an empty bin or scooping with a full bin, since our robot's controller will perform the forces necessary to achieve the given poses.

V. Conclusion

This was just a first step towards the larger area of scooping and kitchen applications for robotics. This can be extended further in the perception and planning problem of determining more precisely how the robot should scoop and how the robot figures out whether it has scooped enough (such as with a feedback loop if pre-planning is insufficient). This could include cameras or other sensors (force, position, weight) that could inform which saved trajectory the robot should execute. Another few key areas are expanding what objects can be scooped and in what quantities. There could be a planning stage of selecting the correct scooper for the desired quantity. Adding deformable or more realistic food-related objects would be a significant step towards the kitchen application. Our scope for this project was limited, but we hope that our work as a preliminary introduction to this topic can set a foundation for future work in scooping.

VI. Team Contributions

The work was split rather evenly, and we largely developed the project during our team work sessions. The sections of this report corresponding to each member's work were generally written by that member. A break down of individual member contributions will follow.

Martin

Martin focused on initial environment setup with the manipulation station and welding the scoop to the iiwa, setting up teleop on the scoop, and making tools for the teleop recording and playback. He also periodically refactored the code to make development smoother and tinkered with parameters (e.g. using the `Sap` discrete contact solver) to improve performance.

Before we began teleop recording and playback, Martin set up the model directives, `MakeManipulationStation`, and pose teleop to check that scooping was feasible. Once that was done, he began setting up our interface to allow saving a series of poses from Meshcat using the slider values. Initially, the system exported poses one at a time to the code block output as executable code that we would manually load in, but he then designed our `SlidersMemory` module and interface to internally track key poses and write to file using a button inside Meshcat.

Martin designed our trajectory file representation as lines of 6-tuples that represent the roll, pitch, yaw, x, y, and z for each pose, but left an interface open for us to decide which poses to automatically save. One of his focuses was producing clean code that was easy for both him and teammates to use and iterate on.

Fiona

Fiona focused on setting up the scoop in the system and developing the geometric scooping approach. Setting up the scoop included creating several sdfs of various size to experiment with which worked well for the other elements of our setup - specifically with the number of spheres and the sizes of the bins. It is also very important that the scoop has a convex collision geometry for ease of simulation. To this end, the mesh $\rightarrow$ sdf converter was used [4].

After system setup, Fiona worked on the geometric approach of creating and commanding the scoop trajectory. During the course of this development, there were several key issues to address. The first was which inverse controller to use: Fiona began with the pseudo-inverse controller from the robot painter notebook but it was no longer able to plan sufficiently when the bins and spheres were added. Fiona looked into other approaches with the differential inverse kinematics controller and adding the scoop to the controller plant. The other main issues were successfully setting up a modified manipulation station that included the scoop in the controller plant, and creating a cup frame in the scoop sdf. Once this was addressed, the geometric approach yielded very promising results, especially from the elliptical path in scooping and circular path in pouring. This work provided a great opportunity to dive deeper into the design process of developing simulations and creating the right path to a solvable problem.

Hannah

During setup, Hannah focused on building and integrating the features that the arm interacted with, such as the bins and objects to be scooped, and ensured that the simulated environment still ran at a reasonable speed. Early on, her focus was largely on integrating custom-built objects and packages with the environment. She created an importable standard environment for use in the many new notebooks created, allowing any environment changes to percolate through all simulations being attempted. After encountering speed and complexity issues associated with simulating and visualising 150+ bodies, she moved the project from scooping rectangular bricks to scooping spheres, which have much less computationally intensive collision geometries.

Afterwards, Hannah's main focus went towards using teleop to learn and then create functional scooping motions. Her work, initially a part of the SlidersMemory class, allows users to pass in a path of desired positions for the iiwa arm to progress through. Hannah built ways to read and use the output from the teleop export tooling Martin had created, so that the arm could "play back" both previous teleop paths and completely new paths. This yielded promising results, and allowed her to develop points for a potential trajectory that would eventually become the successful scooping trajectories presented.

VII. Acknowledgment

We would like to thank the 6.4210 staff for their support throughout lectures, recitations, office hours, and Piazza posts. We would especially like to express our appreciation for the last minute project office hours added in the last week. They were immensely helpful during this last stretch of the project.

VIII. References

A. Links to Code and Demo

Github Repo with objects used in environment setup: github.com/redhann/scooping

Deepnote Workspace with notebooks: https://deepnote.com/workspace/64210-d4bf-68e62101-1cf0-430c-b693-4c10ca0f2662/project/64210-Scooping-0ae32ff5-ed5c-4c6e-a728-d5dde2803bd2/notebook/Guide%20to%20Scooping%20Project-be993360026a4b4a9ec1fefbd9eac6bf. Start with the Guide to Scooping Project notebook for our presentation, videos of scooping, and an explanation of relevant notebooks.

For any questions, please contact the authors at scooping@mit.edu.

References

[1] T. He, S. Aslam, Z. Tong, and J. Seo, "Scooping manipulation via motion control with a two-fingered gripper and its application to bin picking," *IEEE Robotics and Automation Letters*, vol. 6, no. 4, pp. 6394–6401, 2021.

[2] R. Tedrake, *Robotic Manipulation*, 2022. [Online]. Available: https://manipulation.csail.mit.edu/pick.html

[3] O. Grossman. Customizable coffee scoop. [Online]. Available: https://grabcad.com/library/customizable-coffee-scoop-1

[4] Gizatt. Mesh-to-sdf converter. [Online]. Available: https://github.com/gizatt/convex_decomp_to_sdf

[5] R. Tedrake. Authoring a multibody simulation. [Online]. Available: https://deepnote.com/workspace/Drake-0b3b2c53-a7ad-441b-80f8-bf8350752305/project/Tutorials-2b4fc509-aef2-417d-a40d-6071dfed9199/notebook/authoring_multibody_simulation-94c2df76148443f6b3aa76f9fa5a3d32