Sequential Target Pose Planning For Stacking Rigid Body Objects

Bryce Morrow, UNC Chapel Hill

1 Introduction

The ability to autonomously stack irregularly textured rigid body objects is a useful function for many assistive robots to have. Potential applications include home robots that are able to clean a room by organizing objects into categorized stacks or piles (magazines, clothes, etc.) or construction robots that use objects in the environment such as rocks to perform autonomous construction [1]. In this paper, the latter use case serves as our driving inspiration.

There has been increased interest in the use of robots for autonomous construction for the purposes of efficiency and safety. Many construction tasks involve complex movements and back-breaking manual labor, so it is feasible that specialized robots could aid humans with certain aspects of the construction process. To contribute towards this end, we propose a stochastic sequential physics-based pose planning algorithm to find a locally optimal solution to the problem of creating a stack of the largest possible number of objects from a collection of irregularly shaped rigid body objects with known mechanical and geometric properties. The solution to this problem is an ordered sequence of objects and poses, with each pose consisting of a position and an orientation, which fully describes the state of all objects in the stack and consequently, the stack itself.

We also introduce a geometric processing pipeline that takes as input a 3D point scan of a set of objects in an environment and outputs multiple definition files. One of these definition files contains an approximated convex decomposition of the input object and the other includes the full point mesh directly from the environment scan. Having access to both of these object definitions allows us to switch between the two when performing tests in the physics engine. The less precise mesh allows us to trade off speed for accuracy during certain steps of the pose planning process.

2 Related Work

The next best object pose planning algorithm presented in this paper is heavily based on [1]. Both algorithms operate on irregularly shaped rigid bodies and use physics based random sampling to determine locally optimal surface contacts between objects to maximize structural stability.

[2] Explored the construction of ramps using irregular building materials, but utilized an adhesive foam to hold their structures together. [3] Demonstrated an algorithm capable of placing regularly shaped bricks together to build larger structures. Contrary to these approaches, we focus on the case where no adhesives are used and the objects we build with are irregularly shaped.

Other papers such as [5] have explored the use of simulations to perform physics based algorithms based on geometric and dynamic properties of objects. This is a powerful approach for cases where full information about objects in the environment is available, such as in our experimental setup.

Many researchers have explored efficient computational representations of complex rigid bodies. We use the volumetric hierarchical approximate convex decomposition (VHACD) algorithm [6] to simplify some of the computation performed during pose planning.

3 Problem Definition

3.1 Geometric Pre-processing Pipeline

We take as input a 3-dimensional point cloud in the form of an .obj file consisting of all objects of interest in the environment. This information is input to the following pipeline:

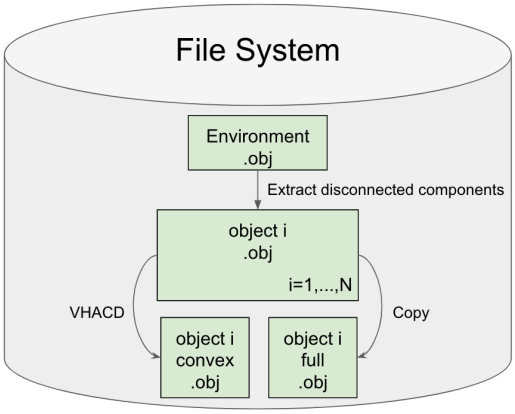

Fig. 1. Visualization of Geometric Preprocessing Pipeline. Green tiles represent objects that exist in the file system as a result of preprocessing. The arrows linking these entities describes the actions performed on the source file to create the destination file.

1) Construct a triangulated environment mesh.
2) Extract all disconnected mesh components from the full mesh as object meshes.

3) Compute an approximated convex decomposition of each object.

The output of this pipeline is two files for each object extracted from the environment mesh: a convex object definition and a full object definition.

3.2 Generating a Sequential Pose Plan

The objective here is to create the largest vertical stack of objects possible (most objects not highest height).

We have a discrete set of objects $O = \{o_1, ..., o_n\}$ where each $o_i \in O$ is defined in our simulation as a point cloud with evenly distributed mass. All o_i are uniquely mapped to some .obj file. Each o_i's position is relative to it's center of mass (COM) and can be represented as

$$(x_i, y_i, z_i) \equiv p_i \in \mathbb{R}^3$$

Every o_i can have some orientation, which we can fully describe as an euler angle

$$(\theta_{xi}, \theta_{yi}, \theta_{zi}) \equiv \theta_i \in [0, 2\pi]^3$$

We consider the joint position and orientation space for all objects $o_i \in O$ as the pose space. The pose space includes all possible configurations of a rigid body object in $\mathbb{R}^3$.

$$\Theta_i \equiv (p_i, \theta_i)$$

To build a stack of objects from the set O, we sequentially select an object o_i and a pose Θ_i. At each iteration, we consider many candidate values for (o_i, Θ_i). For each of these candidates, we compute a cost function that returns high values for bad poses and low values for good poses. We pick the (o_i, Θ_i) that yields the lowest cost across all candidate object pose pairs considered on the current iteration. If we pick the values (o_j^*, Θ_j^*) at the jth time step, and we iterate until we are unable to continue growing the stack, we get a solution of the form

$$\{(o_1^*, pose_1^*), ..., (o_z^*, pose_z^*)\} \text{ for } z <= n$$

We refer to this solution as a pose plan which equivalently represents a stable tower defined by the configurations of its individual components. Objects are sampled without replacement so the pose plan has a maximum length of n since $|O| = n$.

4 METHODS

4.1 Implementation Details

This project is implemented in python using the pybullet physics sdk. The physics simulation is manually stepped as is needed for the pose planning algorithm. All objects have mass (proportional to volume), static friction, and restitution (bounciness).

Running the geometric pre-processing pipeline requires docker and running the experiments conducted later in this paper can be done either through a jupyter notebook or a python script.

The point cloud meshes for the objects were purchased online at CGTrader.com.

4.2 Algorithm for Generating a Target Pose Sequence

The algorithm which generates an ordered sequence of object pose pairs is called the **next_best_pose_search** function. This function relies on the following sub-routines:

- **sample_pose**(obj): randomly sample a position and orientation for an object.
- **valid_pose**($stack, obj, pose$): does the specified object pose result in a stable tower?
- **cost_pose**($stack, obj, pose$): how good is a given valid pose?

4.2.1 *next_best_pose_search*

Input: objects - list of simulation objects

```
n_samples ← 10;
stack ← [];
can_stack ← true;
while can_stack do
    can_stack ← false;
    least_cost ← infinity;
    best_obj ← null;
    best_pose ← null;
    for obj in objects do
        for i = 1, ..., n_samples do
            rand_pose ← sample_pose(obj);
            is_valid_pose ←
              valid_pose(stack, obj, rand_pose);
            if is_valid_pose then
                cost ←
                  cost_pose(stack, obj, rand_pose);
                if cost < least_cost then
                    can_stack ← true;
                    least_cost ← cost;
                    best_obj ← obj;
                    best_pose ← get_pose(obj);
                end
            end
        end
    end
    if can_stack then
        objects.remove(best_obj);
        stack.add((best_obj, best_pose));
    end
end
return stack;
```

We explain the supporting subroutines in detail in the following sections.

4.2.2 *sample_pose(obj)*

Given an input object $o_i \in O$ we sample Θ_i such that the following conditions hold:

- $\theta_i \in \Theta_i$ is sampled randomly and uniformly from the domain $[0, 2\pi]^3$
- o_i's COM is aligned on the z axis with the geometric centroid of the polygon of support of the existing stack.
- When we set the orientation of o_i to θ_i the lowest point in o_i's axis aligned bounding box is ϵ units

above the top of the bounding box of the existing stack. The default value of ϵ we used was .001.

The random sampling of θ_i occurs first. Once this value is set, we enforce the constraints specified by conditions 2 and 3 detailed above. This yields a closed form solution to calculate p_i. We randomly sample an angular orientation as this search space is continuous so it is computationally intractable to consider all possibilities. The alignment of the placed object's COM with the geometric centroid of the polygon of support gives the newly placed object a good chance of settling to static stability after it contacts the top of the tower. We also want to ensure that when the object is set to the sampled position and orientation that it does not collide with any components of the existing tower. This is necessary so that directly after setting the objects pose, we can step the simulation state multiple times and then detect the set of contact points.

4.2.3 *valid_pose*$(stack, obj, pose)$

This routine tests if we can configure an object $o_i \in O$ with a pose Θ_i such that when we step the physics engine multiple iterations forward, the object is statically stable on top of the tower.

To test this hypothesis, we set the object to the desired pose. We disable gravity for the tower and enable gravity for the object. We set the simulation to step forward in a loop. If on any iteration, the COM of the object is outside the x,y bounds of the polygon of support of the current tower, we return false immediately. We maintain a counter of the number of iterations in a row where the object has 3 or more contacts points with the current top of the tower. Once this counter hits some threshold λ (default: 1500), we determine that the object has settled to some equilibrium. Any object will either fail the COM test or hit the continuous contact threshold λ eventually. If the continuous contact threshold is met, we compute the kinetic energy of the object. If this value is below some threshold parameter $E_{k,stable}$ we consider the object stable and return true. Otherwise, we return false. In all cases we disable the object's gravity prior to returning.

4.2.4 *cost_pose*$(stack, obj, pose)$

The cost function indicates how good or bad a particular pose is by:

- **rewarding** a large resultant area for the polygon of support of the newly placed object.
- **penalizing** large kinetic energy values.

To compute the cost of a valid pose we do the following. We set the object to the input pose. We set the collision object for object to be the full definition, rather than the convex approximation. We enable gravity for the stack and for the object. We step the simulation a fixed number of iterations, recomputing the contact point set at each step. This time is meant to allow the object to settle into a stable contact position. Once this happens, we do the following

- we project the contact point set into the x and y dimensions and use this as an approximation of the polygon of support of the object, which we call S_i. We let A_{S_i} represent the area of support polygon.
- we compute the kinetic energy E_{ki} of the object in its contact pose.

The cost is a weighted sum of these two values.

$$\text{cost} = \beta_1 \frac{1}{A_{S_i}} + \beta_2 E_{ki}$$

$$\beta_1, \beta_2 \in \mathbb{R}^+$$

Where all β_i are manually selected parameters. The default values selected for the following experiments were $\beta_1 = 100$ and $\beta_2 = 5$.

Once the cost is computed, we reset the collision object to be the convex approximation.

5 Results

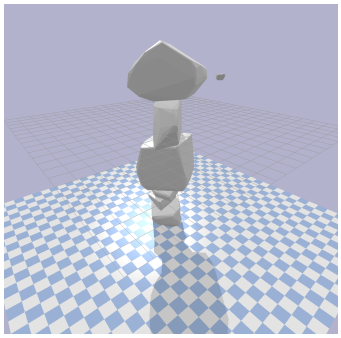

Fig. 2. A stacked tower of size 6, visualized with the openGL rendering bindings for pybullet.

The algorithm proved capable of generating tall stacked towers, albeit inconsistently. I believe this mainly has to do with the simplicity of the cost function, the improvement of which has been left for future work. Below, we detail the average height of stacked tower formed by the algorithm for a set of parameter configurations. In both experiments, for each parameter set, we record the distribution of heights of formed towers over 10 iterations and plot a normalized Gaussian density estimate of this distribution. The main results from these experiments are twofold:

- Increasing the number of random samplings performed per object generally improves performance.
- Increasing the number of objects considered sometimes improves performance but not always. Adding the ability to simulate multiple steps into the future, rather than simply choosing the best object for the current iteration, may help fix this.

6 Conclusion and Future Work

We introduced a greedy sequential target pose planning algorithm which considers the structural stability of various randomized stacking configurations, picking the lowest cost object and pose at each step. A geometric pre-processing pipeline provides us with convex and full object definitions so we can dynamically switch between multiple levels of geometric precision based during the operation of the pose planning algorithm. This framework could potentially serve as an interface which takes input from a real world robot

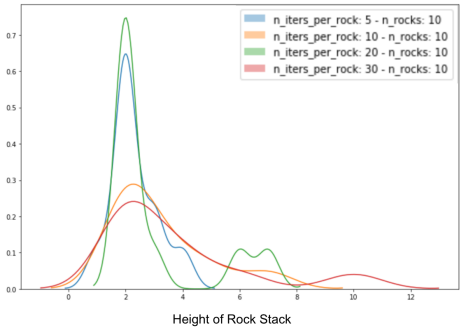

Fig. 3. Holding the number of rocks constant and varying the number of random samplings performed per rock

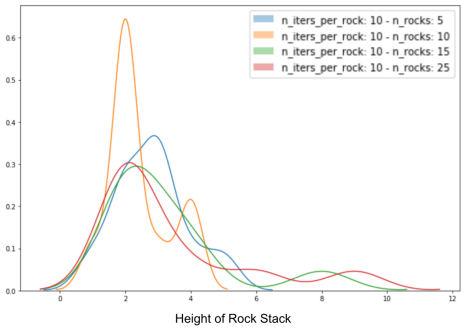

Fig. 4. Holding the number of random samplings per rock constant and varying the number of rocks used.

sensor and outputs a target pose plan to a motion planner which could enable the execution of the sequential pose plan with real objects.

Future work on this framework should focus on improving the quality of the cost function, using a gradient based method to further optimize candidate poses, and simulating multiple steps into the future to account for current decisions impact on future stacking ability.

7 Acknowledgements

This project was completed independently and is not a part of some larger research effort.

References

[1] Autonomous robotic stone stacking with online next best object target pose planning
https://www.research-collection.ethz.ch/bitstream/handle/20.500.11850/254909/1/eth-50701-01.pdf
[2] Distributed Amorphous Ramp Construction in Unstructured Environments
https://cse.buffalo.edu/ nnapp/papers/new.pdf
[3] Mobile Robotic Brickwork
https://www.researchgate.net/publication/300412089_Mobile_Robotic_Brickwork
[4] Perception of physical stability and center of mass of 3-d objects
https://jov.arvojournals.org/article.aspx?articleid=2213254
[5] Simulation as an engine of physical scene understanding
https://www.pnas.org/content/110/45/18327
[6] Volumetric hierarchical approximate convex decomposition
https://www.researchgate.net/publication/327365061_Volumetric_hierarchical_approximate_convex_decomposition