**Abstract**

Recent innovations in construction feature the use of robotic 3D printers to construct buildings with greater speed, efficiency, and affordability than conventional methods. However, these robots are often installed on a large gantry system, limiting the scale of the project being built to what the gantry can reach. Additionally, the sheer scale of the gantry systems make them difficult to move easily and limit their effectiveness. To be effective, particularly in parts of the world where there are critical housing shortages, a robot for construction should be lightweight, robust, and mobile, everything a gantry system is not. One proposal[[1]](#endnote-1)  suggests using a system of two mobile robots instead of a gantry. The authors show that with such a system, the workspace available to the total system is greater than each robot's individual workspace. In this work, I replicate aspects of this paper through simulation in Python Bullet. My emphasis is on writing a path planning algorithm that, upon execution, shows robots constructing a geometric object that is greater than their workspace.

**Description and motivation**

The use of 3D printing robots in construction is rapidly approaching inflection point. The advantages of these robots in construction make them quite desirable, especially since they carry the promise of more affordable final products. The use of 3D printing robots, or the broader category of additive manufacturing has already taken off in other industries, such as in the aerospace and automotive industry.[[2]](#endnote-2) Particularly in the construction industry, which currently accounts for about 30 percent of material waste globally, additive manufacturing will allow for significantly more sustainable projects.[[3]](#endnote-3)

There are a number of research and design firms that have already begun to seriously examine and implement methods for 3D printing . Examples of these are Printhuset,[[4]](#endnote-4) Crane WASP[[5]](#endnote-5), and ICON[[6]](#endnote-6). Unfortunately, the most popular method of 3D printing in construction involve large gantries, which will likely hinder the widespread adoption of the of 3D printing by construction firms. The use of the gantry places significant limits on the size of the building that can actually constructed, as it must be smaller than the gantry itself. This means that the starting height of these 3D printing systems is often at least 10 feet tall. This massive external framework makes these systems difficult to transport and severely limits the sites where they might be used. Additionally, these systems often feature only one print head, limiting the efficiency of printing.[[7]](#endnote-7)

It is particularly important that these systems are refined and improved because of the enormous promise they hold in solving the global problem of affordable housing. The world is urbanizing rapidly[[8]](#endnote-8), creating a massive demand for housing that in many places goes unmet due to lack of efficiency in home construction, which leads to an increase of house prices. In many of the world’s largest cities, housing costs range from 200 to 3000 percent of incomes, pushing many to the periphery.[[9]](#endnote-9) Currently 25 percent of the world’s urban populace live in informal settlements, which are characterized by uncertainty and lack of economic opportunity[[10]](#endnote-10). The United Nations has recognized the importance of this issue, marking it as their 11th Sustainable Development Goal for 2030: making cities and human settlements safe, resilient, and sustainable.[[11]](#endnote-11)

Systems for 3D printing homes, if made widely available and affordable for use by construction firms, might create a large dent in this problem. However, in order for this to happen, it is critical to move away from gantry-based systems, and towards more scalable solutions. Zhang et al. offer one such solution. They have explored a proposal for concurrent 3D printing by a team of mobile robots. They argue that their system, taking advantage of “localization, collision avoidance, and efficient coordinated printing through optimal robot placement” demonstrates the needed scalability by allowing the individual robots to print “single piece structures of arbitrary sizes” [[12]](#endnote-12). This is a particularly interesting method, that takes advantage of many concepts in robotics to create a necessary advancement in 3D printing technology.

In this project, I extend their application by simulating a few aspects of what they have done, and also what they noted as possible areas for future exploration. For example, they note that the robots could be made even more efficient if programmed to move and print at the same time, as opposed to staying in one position.

The remainder of this paper is organized in the following way. In Section 2, I explain the algorithms I considered when implementing this project. In Section 3, I describe the process of implementing the simulation, and results I generated along the way. In Section 4, I present my reflections and analysis.

**Approach**

In working through this project, my goal was to develop a better understanding of how 3D printing might work at a larger scale through creating a simulation. This was a somewhat open-ended goal, so I began by thinking about what aspects of the project related to some of the concepts we had studied in class. I thought that path planning, trajectory planning, and coordination would be the most critical aspects to consider.

*Path Planning*

For path planning, I wanted to get a better understanding how conventional 3D printers analyze paths and plan geometries in the status quo. For this, I looked at Sheng et al.’s[[13]](#endnote-13) paper which suggests a decomposition approach for tool planning with 3D printers. They note that in order to understand the geometric structure of a free form surface, for example a car door, it is sufficient to analyze its 2D projection. They then further segment this 2D shape into sub-polygons using what the identity as “good partitions”. These “good partitions” are characterized by their rectangular properties. The goal of their method is to divide the geometry into shapes that will be easy for the tool to accommodate.

*Trajectory Planning*

Thompson and Yoon[[14]](#endnote-14) also consider the path planning problem, however, with a focus on increasing efficiency through trajectory planning. They determine optimal trajectories for an arbitrary additive manufacturing tool working in a planar space. They suggest a combination of two motion control methods: linear segments with parabolic blends (LSBP), and minimum time trajectory (MTT). In the LSBP method, they acknowledge that a tool following a given path should do so at a constant velocity, but can ramp up to and down from those velocities, creating a trapezoidal velocity profile. The MTT method is meant to minimize the transition time between two path segments, through constant acceleration and then deceleration as the path segment is approached. They showed that the combination of these two control methods allow for a 3D printing tool to complete its job more efficiently.

*Robot Coordination*

Zhang et al[[15]](#endnote-15) considers a number of issues relating to path planning, localization and coordination. In order to determine optimal configurations of the robot arm, given the desired path and the presence of the other robot, Zhang adopts a kinematic reachability algorithm. For each configuration, they evaluate the reachability of a given robot printer with regards to the goal state as well as the cost from a collision between the robots. They are then able to choose the configurations that maximize this evaluation. They also use GMapping (a popular solution to the Simultaneous Localization and Mapping problem) coupled with data from sensors on board their robots to create a map of the robot’s “world” and its position in that world.

*Summary*

I found that as I implemented my project, while I initially planned to implement a few of the methods found above, the process of building the simulation from the ground up meant that I ended up focusing on other aspects. For example, I spent a lot of time thinking about how to actually path plan for a mobile robot along continuous segments. I was unable to find many papers that discussed this problem in detail. Existing 3D printers for construction are often really just large scale 3D printers, with a single tool moving around a gantry, where the robot has a very clear understanding of the world in which it operates. This was not necessarily the case for the simulation I was working on.

**Implementation**

*PyBullet Environment*

Bullet Physics is a professional open source collision detection, rigid body and soft body dynamics library written in C++.[[16]](#endnote-16) PyBullet, built on Bullet Physics, is a fast and easy to use Python module for robotics simulation and machine learning.[[17]](#endnote-17) I chose to use PyBullet as my simulation environment as we had been introduced to it during a lab, and I thought its capabilities for easy integration with machine learning libraries would make it useful to be familiar with for future projects. I chose to use Python as opposed to MATLAB because it is an open source platform, meaning that the final simulation would be easier to share once fully completed. An interesting use for this project would be an online website where one could input whichever geometry they liked and see any number of robots working to build the geometry. Although that is out of the scope of the project, this desire drove my decision to use Python. Additionally, as I was working through the project, I found Pandas[[18]](#endnote-18), a Python library for data analysis to be very helpful.

*Initial Steps*

Although I knew that I eventually would want to move the project to a fully Python environment, I began working on the simulation in MATLAB. This wass because I found a really helpful tutorial about shape-tracing on the MATLAB website, which I followed to get a better understanding of the process.[[19]](#endnote-19) From working through this tutorial, I was able to get a clearer idea of what steps I would need to take in order to simulate Zhang’s project. The first was turning a geometry into a set of waypoints for the robot to follow. The MATLAB example uses the desired shape of the MATLAB membrane logo, and divides this shape geometry across x-y, y-z planes. I chose to use a cylinder since it had a constant shape in the x-y plane, and I could easily move forward with the tutorial by only varying the z heights for the robot. I also chose to focus on this approach of splitting the geometry based by elevations profiles because it seems to be more widely used in industry.[[20]](#endnote-20) Figure 1a.

Once I extracted these waypoints, I visualized them to ensure they were within the robot’s workspace. I then passed the points through an inverse kinematics solver to get the configurations for the robot arm. I then passed the trajectories through a Polynomial Trajectory Block to create a smooth trajectory for the arm to follow. The benefits of this are that one can limit any abrupt changes in the robot’s acceleration. I found the actual trajectory taken by the end effector by passing the configurations through a forward kinematics block and plotting this. Figure 1b.

*Determining Paths For an Arbitrary Geometry*

After going through the MATLAB tutorial, I had a much clearer idea of what I had to do for the project. The first step required extracting paths for arbitrary 3D geometries. I worked in Google Colaboratory[[21]](#endnote-21) for this aspect of the project since the interface makes data visualization really fast and easy.

First, I had to generate an arbitrary 3D geometry using matplotlib[[22]](#endnote-22), a Python library for data visualization. Next, I used the contour() function within matplotlib to essentially slice up the geometry into 2D shapes at different z levels. After much trial and error, I determined how to use the contour() function to create an object with a parameter allsegs() that has the x,y points organized by continuous blocks segments at a given level. Once I understood this, I could reformat the data as a Pandas DataFrame to be organized by x,y,z points, paying no attention to whether the points formed contiguous segments. See Figure 2a,b,c,d.

*PyBullet Simulation of Geometry in Workspace*

I could now set up a simulation in PyBullet. Working from this example (inverseKinematics.py)[[23]](#endnote-23), I was able essentially replicate the work that I had done in MATLAB. I chose to use the Kuka Iiwa robot because it was lightweight, mobile and designed for human interaction.[[24]](#endnote-24)

For the first iterations, I focused on geometry that was constrained to the robot’s workspace. I decided what the robot’s workspace would be by testing the limits of the end effector in the simulation and noting them down. See Figure 3 for the determinations I came to. These are similar to the limits noted in the robot’s specification.[[25]](#endnote-25) See Figure 3. Armed with this knowledge, I was able to create geometries that were within the x, y, and z bounds of the robot workspace only. I passed these through an inverseKinematics solver provided by the PyBullet library to determine which joint configurations the geometry paths would require. I executed the configurations using a setJointMotorControl tool also within the PyBullet library. I used depugPaths, yet another tool within the the PyBullet library, to visualize the path taken by the end effector, and ensure it was following the correct trajectory. See Figure 4.

*PyBullet Simulation of Geometry Greater than the Workspace*

Once I was sure that the simulation was working for geometries in the workspace, I then turned to the task of determining how the robot would handle shapes that were greater than its workspace. This would require the robot to move. In order to move intelligently, the robot would need to know where it was relative to the geometry it was trying to print.

In accomplish this goal, I first acknowledged that with two robots working at a time, the easiest way to split up the work would be along either the x or y axes. I decided to split the geometry along the y axis, although it could have been split along the x-axis without issue. One robot would tackle the half of the geometry with y’s greater than 0, and the other robot would handle geometries with y’s less than zero. This simplified the problem, as I now only had to worry about the robot travelling around what would, for an arbitrary geometry, be roughly approximated to a semi-circle.

As it was possible for me to determine the robot’s position in the simulation directly by querying with the getLinkState tool in PyBullet, I did not find it very useful to create a total map of the environment as discussed in Zhang’s paper.[[26]](#endnote-26) Instead, I needed only for the robot to understand where it was relative to the geometry it was working on printing. To accomplish this, I created a bounding box in front of the robot that denoted its workspace. I called this the robot bounding box, or rBB. The purpose of creating the bounding box was to ensure that, for a given position in the world space, the robot will only be asked to “print” points that are in its workspace, and to avoid unreal solutions out of the inverse kinematics solver.

I then identified the furthest point within the geometry from the robot, that is also at the z= 0 plane, and placed an “object” bounding box there which aligned with the furthest corner. I chose the furthest point in the geometry because the robot should ideally navigate from left to right, then right to left, as it handles different elevations. I choose to start with the z=0 plane considering the application of this work in 3D printing, where a base is built upon by adding material at higher heights. The object bounding box and the robot bounding box share the same dimensions.

I then navigated the robot to a position where the centroids of the two bounding boxes were aligned. I checked which of the points in the geometry at the z= 0 are within this bounding box, and then determine what joint configurations the robot will have to cycle through in order to have its end effector go through these points. This is done using the same method as a if the robot were printing the geometry inside its workspace, as discussed earlier. The robot executes these trajectories and then returns back to a rest orientation.

Next, the robot must identify where it should go in order to continue printing the geometry. To do this, I asked the robot to forget the points that is had just executed within the z=0 plane. I then created a “guess” point that is one bounding box length away from the midpoint of the current object bounding box. I then determined which point in the remaining geometry was closest to this “guess” point. This point is made the centroid of a new object bounding box. (Note that the initial bounding box has the identified point located in far corner, as opposed to the center). This is done so that as many points as possible will fall into the bounding box, decreasing the number of total movements the robot must make. I then navigated the robot until it’s personal robot bounding box is aligned with the new object bounding box, and “prints” the points within this space. The robot repeats this process until it has “forgotten” all the points at the z=0 level. It then moves to the subsequent z level and repeats the process until it has gone through all the points in the geometry. Figure 5.

**Reflection and Analysis**

Through this process of working on this simulation, I learned about different methods for path planning, and a few of the complexities that can go into creating an optimal trajectory. A way to improve this simulation would be to incorporate the time-dependent trajectories discussed by Thompson and Yoon. This would allow for the robot to print the geometry in a shorter time. However, the algorithm they present would have to be modified to consider the trajectory of the robot itself, and the trajectory of its end effector.

I also learned a little about robot coordination. I ultimately implemented a very simple coordination strategy. I considered that the geometry should be split and each robot given its own share to print. Since the total size of the geometry is larger than the robot’s workspace, it is easy to simply have the robot’s begin from opposite ends of the geometry so they never collide. A more robust implementation of path coordination would combine Zhang’s coordination approach (discussed in Section 2) with the decision of where the robot should go next.

I felt that going through the unit on path planning and obstacle avoidance really helped me as I read through papers on the subject. Also, understanding kinematics and inverse kinematics was really helpful, even though I did not implement those directly. I would have liked to spend time discussing path planning for mobile robots in the course, as I feel that would have helped with this project.

|  |  |  |
| --- | --- | --- |
| Workspace Limits for Kuka Iiwa Simulataion in PyBullet (OpenGL Render) | | |
| x | -0.7 | 0.7 |
| y | -0.7 | 0.7 |
| z | 0 | 1 |

Sources

1. Zhange et al [↑](#endnote-ref-1)
2. <https://markforged.com/blog/five-industries-utilizing-3d-printing/> [↑](#endnote-ref-2)
3. https://connect.bim360.autodesk.com/robotics-in-construction-history [↑](#endnote-ref-3)
4. Printhuset [↑](#endnote-ref-4)
5. Crane WASP [↑](#endnote-ref-5)
6. ICON [↑](#endnote-ref-6)
7. Zhang et al [↑](#endnote-ref-7)
8. Source about world urbanizing rapidly [↑](#endnote-ref-8)
9. The Global Housing Crisis, Citylab, 2018 [↑](#endnote-ref-9)
10. The Global Housing Crisis, Citylab, 2018 [↑](#endnote-ref-10)
11. <https://sustainabledevelopment.un.org/sdg11> [↑](#endnote-ref-11)
12. Zhang et al [↑](#endnote-ref-12)
13. Part Geometric Understanding for Tool Path Planning in Additive Manufacturing, Sheng et al, 2003 [↑](#endnote-ref-13)
14. Efficient Path Planning Algorithm for Additive Manufacturing Systems [↑](#endnote-ref-14)
15. Zhang et al [↑](#endnote-ref-15)
16. <https://github.com/bulletphysics/bullet3/blob/master/docs/Bullet_User_Manual.pdf> [↑](#endnote-ref-16)
17. [https://docs.google.com/document/d/10sXEhzFRSnvFcl3XxNGhnD4N2SedqwdAvK3dsihxVUA/edit#](https://docs.google.com/document/d/10sXEhzFRSnvFcl3XxNGhnD4N2SedqwdAvK3dsihxVUA/edit) [↑](#endnote-ref-17)
18. <https://pandas.pydata.org/pandas-docs/stable/index.html> [↑](#endnote-ref-18)
19. <https://www.mathworks.com/help/robotics/examples/manipulator-shape-tracing.html> [↑](#endnote-ref-19)
20. Sheng et al [↑](#endnote-ref-20)
21. <https://colab.research.google.com/notebooks/intro.ipynb> [↑](#endnote-ref-21)
22. <https://matplotlib.org/index.html> [↑](#endnote-ref-22)
23. Inversekinematics thing [↑](#endnote-ref-23)
24. <https://www.kuka.com/en-us/products/robotics-systems/industrial-robots/lbr-iiwa> [↑](#endnote-ref-24)
25. <https://www.kuka.com/-/media/kuka-downloads/imported/6b77eecacfe542d3b736af377562ecaa/db_lbr_iiwa_en.pdf?rev=013dce5b46bd4137aa21c50f82e912dd&hash=52CF1F3011FC69677750841C4E7DFE4D> [↑](#endnote-ref-25)
26. Slam?

    Figures [↑](#endnote-ref-26)