

A Slicer and Simulator for Cooperative 3D Printing

Jace McPherson, Adam Bliss, Flora Smith, Edmund Harriss, Wenchao Zhou
The AM³ Lab, Department of Mechanical Engineering, University of Arkansas at Fayetteville
Fayetteville, AR, United States of America; Corresponding email: zhouw@uark.edu

Abstract

Cooperative 3D printing is an emerging technology that aims to increase the 3D printing speed and to overcome the size limit of the printable object by having multiple mobile 3D printers (or printhead-carrying mobile robots) work together on a single print job on a factory floor. It differs from traditional layer-by-layer 3D printing due to the requirement for multiple mobile printers working simultaneously without interfering with each other. Therefore, a new approach for slicing the CAD model and generating commands for the mobile printers is needed, which has not been discussed in the literature before. We propose a chunk-by-chunk based slicer that divides an object into chunks so that different mobile printers can print different chunks simultaneously without interfering with each other. In this paper, we developed a slicer for cooperative 3D printing with two mobile fused deposition modeling (FDM) printers. In order to validate our slicer and visualize the cooperative 3D printing process, we have also developed a simulator environment, which can be a valuable tool in optimizing cooperative 3D printing strategy. Results show that the developed slicer and simulator are working effectively.

1. Introduction

Although additive manufacturing (AM) has become increasingly popular in recent years, it has been significantly limited by its slow printing speed and the size of the object it can print. Cooperative 3D printing is an emerging technology that aims to address these limitations by having multiple printhead-carrying mobile robots (or mobile 3D printers) work together on the same print job on a factory floor. With the traditional layer-by-layer 3D printing approach as defined by the ASTM F42 committee [1], it would be difficult for the mobile 3D printers to cooperate without interfering with the already printed part and with each other, which calls for a different approach of 3D printing.

In traditional 3D printing, a CAD model needs to be sliced into layers and the path of the printhead movement needs to be planned to deposit materials for each layer. A slicer usually works by intersecting a plane at different Z-heights with the CAD model and calculating the boundary segments on each layer. The movement path of the printhead is then determined to infill the region within the boundary at each layer. Many different slicers have been developed, such as Slc3r [2], Cura [3], Kisslicer [4], and Skeinforge [5]. C Kirschman et al. has developed a parallel slicing algorithm to improve the slicing speed [1, 6]. E Sabourin et al. presented an adaptive slicing algorithm for layer-based 3D printing that can slice with different layer thickness [7]. S. Lefebvre et al. reported a GPU accelerated slicer [8]. However, slicing for the emerging cooperative 3D printing technology has not been investigated before.

A slicer is usually accompanied by a visualizer for the user to see the slicing results. A G-code viewer is one of the most common visualizers, such as the built-in viewer in Repetier Host

[9]. Because cooperative 3D printing involves multiple robots, a simulator that can visualize the dynamic path of each mobile robots and how the materials are deposited over time will be beneficial for validating the printing path and optimizing the printing strategy for cooperative 3D printing. Many different simulators have been developed for mobile robots, such as Gazebo [10], EyeSim [11], UberSim [12], and Simbad [13]. These robot simulators can effectively simulate the interaction of multiple robots in 2D or 3D for evaluation of the design and the behavior of the robots. However, simulators for visualizing the dynamic 3D printing process of mobile 3D printers have not been reported.

In this paper, to address the possible geometric interference arising from the layer-by-layer based approach with multiple mobile 3D printers, we present a chunk-by-chunk based slicing approach so that each mobile 3D printer only needs to print a small chunk at a time, which can effectively separate the mobile 3D printers. The chunk-based printing can also keep 3D printing localized and therefore potentially avoid the large temperature gradient and internal stress that are common with 3D printing large objects. With proper scheduling of each individual mobile printer, this approach can be scaled to a very large number of mobile printers without interference. To simplify the problem, the slicing algorithm in this paper will be limited to the cooperative 3D printing between two mobile 3D printers that carry a fused deposition modeling (FDM) extruder. This chunk-by-chunk based slicing algorithm ensures good bonding between the chunks and smooth transitioning when a mobile robot moves from one chunk to another. To visualize the dynamic cooperative 3D printing process, we have also developed a simulator that can simulate the dynamic printing process over time and the communication between mobile robots. The output of the slicer can be directly passed to the simulator for visualizing the dynamic printing process and can also be readily translated into hardware commands for real mobile printers to print. Our results show that our slicer works effectively with the simulator in evaluating the multi-robot printing strategy. This new slicer is an essential part of the emerging cooperative 3D printing technology and the simulator will provide a valuable tool for us to understand and optimize the printing strategies for cooperative 3D printing.

This paper is organized as follows. A discussion of the overall slicing process is presented in section 2. Section 3 presents the development of a chunker for the chunk-based slicing. The slicer is discussed in section 4. Section 5 presents a simulator to simulate the sliced results. Conclusions are given in section 6 and future work in section 7.

2. Slicing for Cooperative 3D Printing

At the core of the cooperative 3D printing platform is a mobile 3D printer as shown in Figure 1, which replaces the XY stage on a regular 3D printer with a set of omnidirectional wheels to translate the printhead in XY direction. This design enables unlimited printing in the X direction, but the Y direction is limited by the distance between the printhead and the front wheels (termed as “build depth” in this paper) if a layer-by-layer based approach is used because the printed material in the previous layers will block the path of the wheels in Y direction.

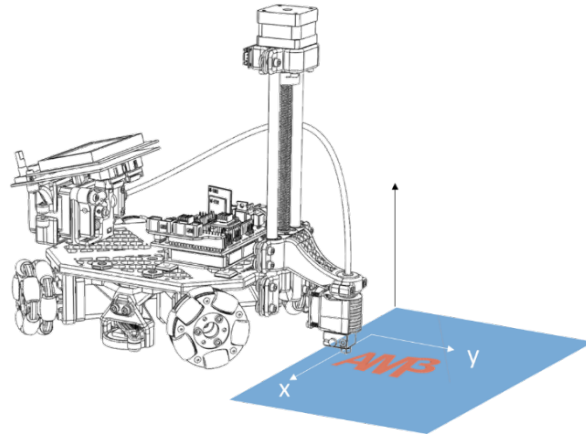


Figure 1. Illustration of a mobile 3D printer: it can print indefinitely in X direction but limited in Y direction

In this paper, we propose a chunk-by-chunk based printing strategy, where the mobile printer finishes all the layers of one chunk before it moves to print another chunk, effectively solving the problem of the blocked path by the printed materials to enable the mobile printer printing unlimited in both X and Y directions. One issue that arises is the bonding between the chunks. Our solution to this issue is to use a sloped interface (and/or an angled printhead) to allow more bonding surface between the chunks. A general slicing strategy for cooperative 3D printing is illustrated in Figure 2:

1. Chunker: A CAD model of the print job will be first input into a “chunker”, which splits the CAD model into chunks based on a set of criteria to ensure feasible printing of each chunk and good bonding between chunks.
2. Slicer: The chunks will then be sliced into layers using a slicer, which generates commands for printing the chunks (e.g., tool paths, material extrusion, temperature control, etc.), schedules the sequence of printing the chunks among multiple robots, and inserts communication commands to enable necessary communication among multiple robots.
3. Simulator: The commands generated by the slicer are interpreted by a simulator, which visualizes and animates the dynamic printing process over time to provide a tool for evaluating the chunking and slicing parameters and results.

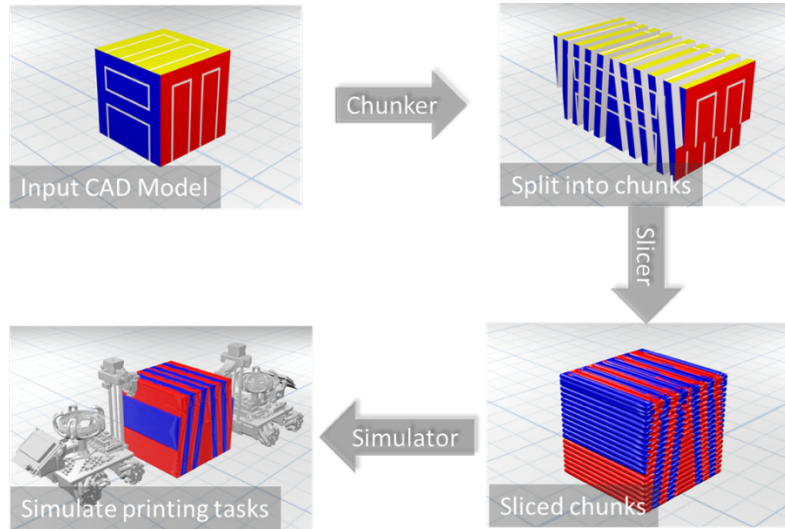


Figure 2. Illustration of the slicing strategy for cooperative 3D printing: (1). The chunker splits the printing job into chunks and ensures feasible printing of each chunk and good bonding between chunks; (2). The slicer slices the chunks into layers, generates commands for printing the chunks, schedules the sequence of printing the chunks among multiple robots, and inserts communication commands to enable necessary communication among multiple robots; (3). The simulator visualizes the dynamic printing process using the commands generated by the slicer.

3. Chunker

The objective of chunking is to divide the printing job into chunks such that they can be assigned to as many robots as possible to increase the printing speed. Therefore, the overall chunking strategy is highly dependent on the geometry of the print, the number of available robots, and how the robots will be scheduled. To simplify the problem, we will only consider how to chunk for two robots in this paper and leave the scaling to many robots for the future. The methodology for splitting a print job into chunks for two robots will generally be applicable for many robots through a “divide and conquer” strategy.

To chunk for two robots, we will split the object into multiple chunks along one direction (Y direction in Figure 1) with sloped planes to ensure good bonding between chunks. Two robots start from the center chunk and print along +Y and −Y direction respectively to finish each chunk. To calculate the geometries of these chunks, we simply bisect the original geometry multiple times around multiple planes. Because we have constrained the problem to chunking only in the +Y and −Y directions, each plane can be defined by two parameters: slope and Y-position.

3.1. Slope Determination

A sloped interface between chunks is needed for this chunk-by-chunk based 3D printing strategy. The angle of the sloped plane needs to be carefully determined due to conflicting objectives:

1. A maximum slope angle will maximize the volume of each chunk and increase printing efficiency;
2. A minimum slope angle will maximize the area of the bonding interface and increase the bonding strength.

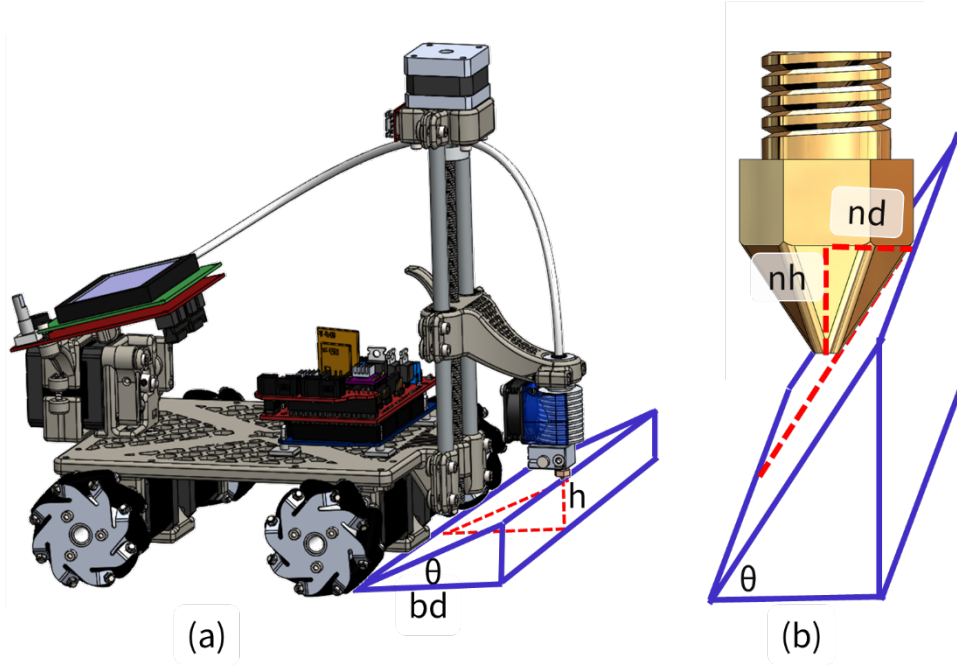


Figure 3. Illustration of robot build limits: (a) The smallest slope angle of a chunk depends on the ratio of the object height, h , to the robot build depth, bd ; (b) The largest slope angle of a chunk is limited by the ratio of the nozzle height, nh , to the nozzle depth, nd .

In addition, the range of the slope angle is limited by the robot parameters as illustrated in Figure 3, which should be determined by:

$$\theta_{max} = \tan^{-1} \left(\frac{nh}{nd} \right) \quad (1)$$

$$\theta_{min} = \tan^{-1} \left(\frac{h}{bd} \right) \quad (2)$$

where θ_{max} and θ_{min} are the limits of the slope angle, nh and nd are the nozzle height and nozzle depth, h , is the height of the object to be printed, and bd is the build depth of the printer.

If the angle is too large or too small, either the front wheels of the robot or the nozzle will interfere with the printed material. It should be noted that the range of the angle is dependent on the printer design and the limits can be easily changed with a tilted nozzle or a printer with an adjustable build depth. Real-world tests should be performed to choose an appropriate slope angle. In this paper, we use the calculated θ_{max} for all our tests.

3.2. Chunking Plane Determination

With a determined slope, we will also need to know where we want to split the object. For the chunking strategy with two robots, we first need a center chunk, which can only be printed by one robot. After the center chunk is completed, the two robots will finish the chunks on the left and the right side respectively. The center chunk's chunking planes can both be

represented with their normal vector, \underline{n} , and any point on the plane, \underline{p}_p . This is the most convenient representation because the bisecting algorithm we use (specifically, the bisecting algorithm in Blender) requires only these two values to bisect a 3D geometry around the plane. The left and right chunking planes for the center chunk can be determined by:

$$\text{Plane L: } \underline{n} = \left(\underline{c} \times \left| (0,0,h) + \frac{h}{\tan(\theta)} \perp \underline{c} \right| \right); \quad \underline{p}_p = (\underline{p}_c + \frac{h}{\tan(\theta)} \perp \underline{c}) \quad (3)$$

$$\text{Plane R: } \underline{n} = \left(\underline{c} \times \left| (0,0,h) - \frac{h}{\tan(\theta)} \perp \underline{c} \right| \right); \quad \underline{p}_p = (\underline{p}_c - \frac{h}{\tan(\theta)} \perp \underline{c}) \quad (4)$$

where \underline{c} is the normal vector of the center line of the object, \underline{p}_c is a point on the center line, θ is the angle of the chunking plane previously determined, and

$$\perp (x, y, z) := (-y, x, z) \quad (5)$$

After calculating these two planes, we can iteratively shift those planes outward by a shift amount, s , from the center chunk (by iterating $\underline{p}_{i+1} \leftarrow \underline{p}_i + s\underline{c}$). We can use these planes to bisect the model into subsequent “left” and “right” chunks. Figure 4 demonstrates the iterative chunking process, starting with the center chunk, and then shifted the chunking planes to the right and the left respectively.

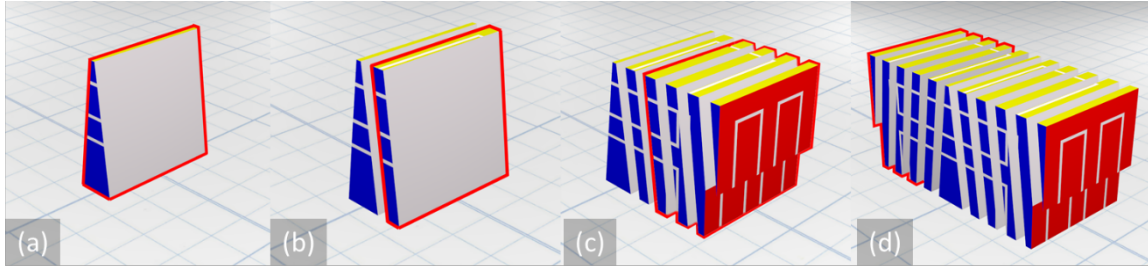


Figure 4. Iterative chunking results. Planes L and R are reused and shifted to split further chunks on the left and right of the center chunk: (a) center chunk; (b) shifted plane R to the right by one chunk; (c) shifted plane R to all the right chunks; (d) shifted plane L to all the left chunks.

We have applied this chunking algorithm to two different geometries using different chunking settings to demonstrate its effectiveness, including a cylinder and a car model, as shown in Figure 5. The yellow chunk is the center chunk. As we can see, the chunker works effectively with complex geometries and different settings.

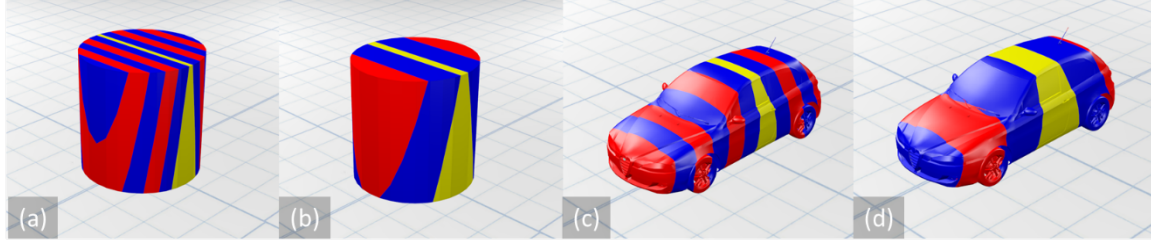


Figure 5. Chunker results with two different objects, a cylinder and a car model: (a) Cylinder with short build depth and steep chunking slope; (b) Cylinder with deep build depth and moderate slope; (c) Car with short build depth and steep chunking slope; (d) Car with deep build depth and moderate slope.

4. Slicer

The objective of the slicer is to make sure the robots can work together to finish the printing according to the printing strategy. Unlike a regular slicer that only generates the tool path, the slicer for cooperative printing needs to accomplish three functions:

1. Assign chunks to each robot and determine their printing sequence;
2. Generate tool paths for each chunk and the tool paths for transition between chunks;
3. Generate commands based on the tool path for the robots to execute and provide a mechanism for the robots to communicate with each other in case one robot's printing task is dependent on the status of the printing task of another robot.

4.1 Printing Sequence

In order to determine the path for a robot to follow, the robot must first know the chunks it will print and their sequence. As we only consider two robots in this paper, we can use the following simple strategy to assign the chunks to the robots, where C_A represents Robot A's chunks and C_B represents Robot B's chunks:

$$C_A = [\text{center chunk}, \text{left chunk 1}, \text{left chunk 2}, \dots] \quad (6)$$

$$C_B = [\text{right chunk 1}, \text{right chunk 2}, \dots] \quad (7)$$

where Robot A is assigned with the center chunk and all the chunks on the left, and Robot B is assigned with all the chunks on the right. The chunks then need to be ordered based on the scheduling strategy for the print job. Because the chunks were generated in order by the chunker, there is no need to order the chunks for the simplified two-robot printing in this paper.

4.2 Tool Path Generation and Transition between Chunks

With the ordered chunks assigned to each robot, we need to generate a sequential tool path for each robot to finish its assigned chunks. This task can be accomplished in steps as illustrated in Figure 6: (1) Generate tool path for each chunk; (2) Generate tool path between chunks; (3) Combine the tool paths in sequence.

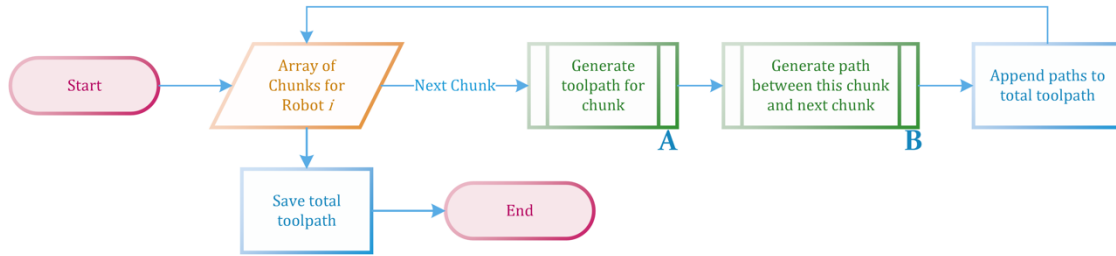


Figure 6. Path generation for each robot: generate tool path for each chunk and generate path to transition between chunks

4.2.1 Tool Path Generation for Chunks

The tool path generation for a chunk is similar to what a regular slicer does for a printing object. The general process is illustrated in Figure 7. Instead of starting from an STL file (or AMF or other file formats), the tool path generation algorithm starts with triangular meshes generated by our chunker. Based on the specified layer thickness, a list of horizontal planes is generated to split the model into multiple layers. The horizontal planes are then intersected with the triangular mesh to calculate the intersection line segments at each layer. Figure 8 shows an algorithm we used to calculate the line segments at each layer. The line segments are then ordered into a ring to form a perimeter for each layer. Infill path are then generated for the parameters at each layer. Because this process has been well established in current slicers, we are omitting the details here.

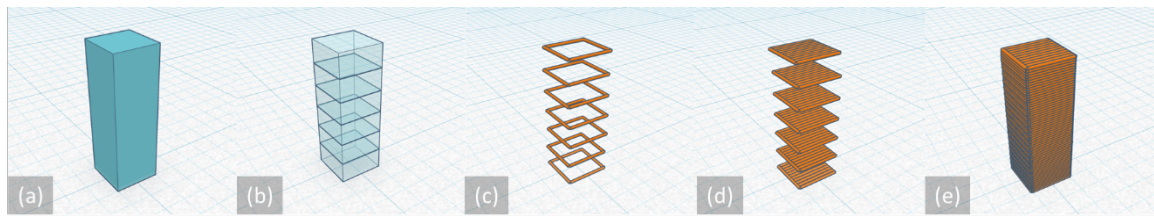


Figure 7. Illustration of the tool path generation process: (a) Original model; (b) Model split into horizontal layers; (c) Perimeters calculated for each layer by intersecting the triangular mesh with a plane at each layer; (d) Generate infill path for the perimeter at each layer; (e) Combine all the tool path generated at each layer.

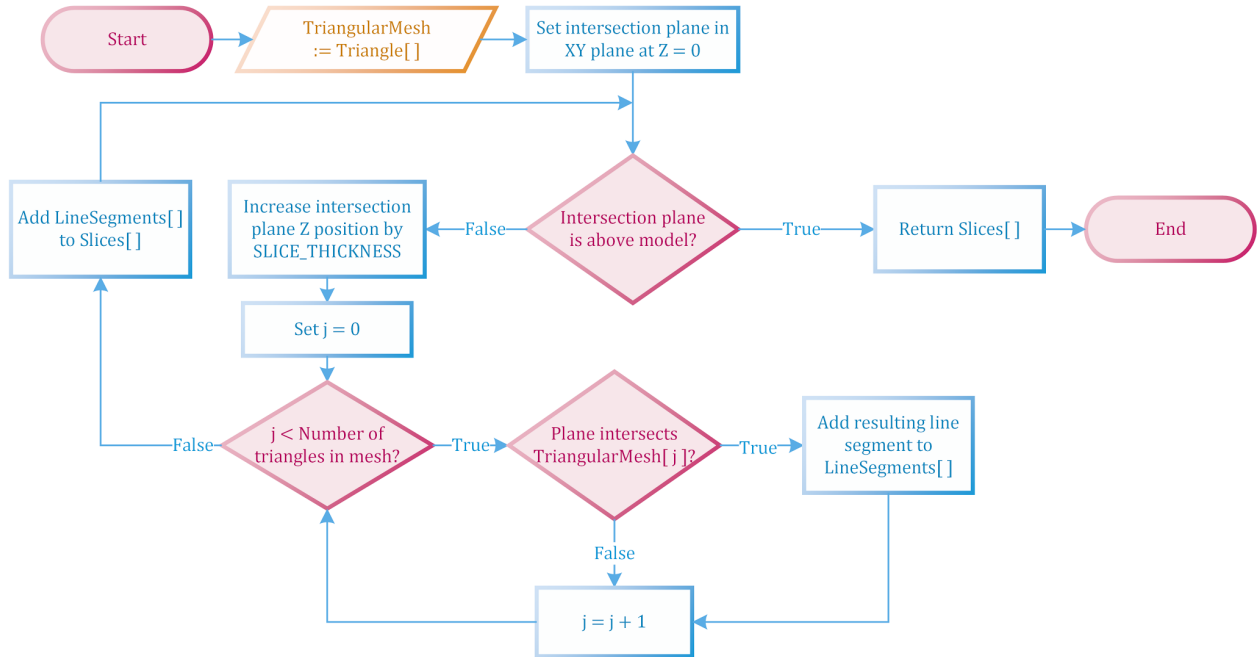


Figure 8. Process for calculating the line segments for the perimeter path of a layer.

The results of the tool path generation can be seen in Figure 9, which shows the slicing and infill algorithms are working correctly.

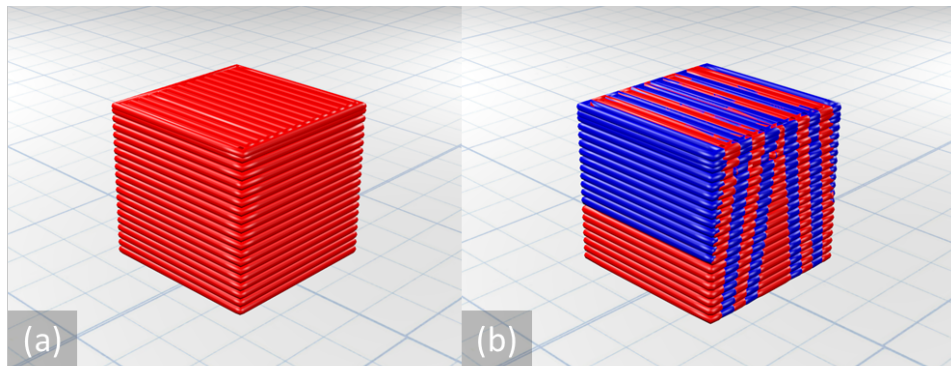


Figure 9. Tool path generation from the slicer algorithm (thick filament was used for better visualization): (a) without chunking; (b) with chunking first

4.2.2 Transition between Chunks

In addition to the print path for each chunk, the robot must have a way of moving from one chunk to the next. A simple direct line would not work, as the robot could knock against previously printed materials. Instead, we generate a separate path from the endpoint of the current chunk to the starting point of the next chunk. There are ways to optimize this path to save printing time, but we are using a simple approach that is not optimized but always works. Figure 10 visually demonstrates how we generate this transition path. Four points comprise the path: the endpoint of the current chunk $p1$, the start point of the next chunk $p4$, and two points in between. For $p2$, we simply shift the printhead upwards a small amount. For $p3$, we move the extruder to

the boundary of the chunk (i.e. to the same x and y position as $p4$). The path generation process is detailed in Figure 11.

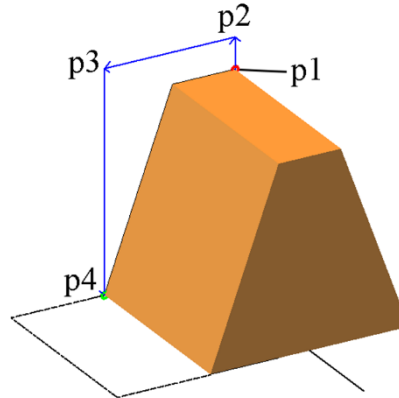


Figure 10. Transition path between chunks ($p1 \rightarrow p2 \rightarrow p3 \rightarrow p4$): the printhead moves slightly upward from previously printed materials and navigates to the next chunk. The endpoint of the current chunk is $p1$ and the start point of the next chunk is $p4$. $p2$ and $p3$ are points used to generate the transition path.

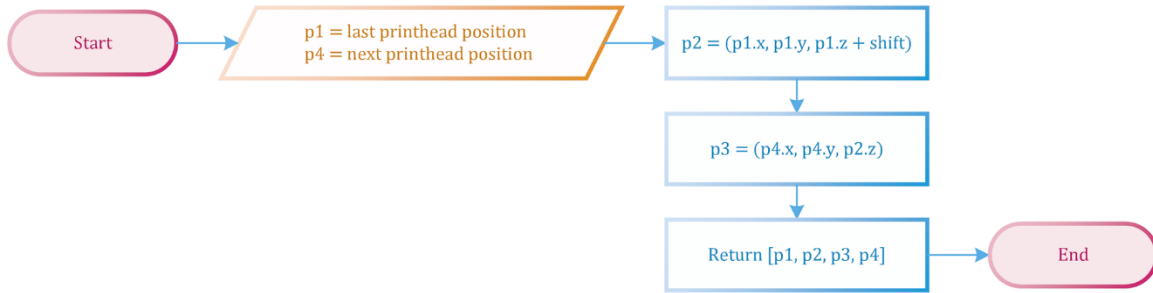


Figure 11. Process of generating transition path between two chunks

4.3 Command Generation and Communication

At this point, we have generated the entire tool path for each robot, which is organized in a multi-level hierarchical data structure as illustrated in Figure 12.

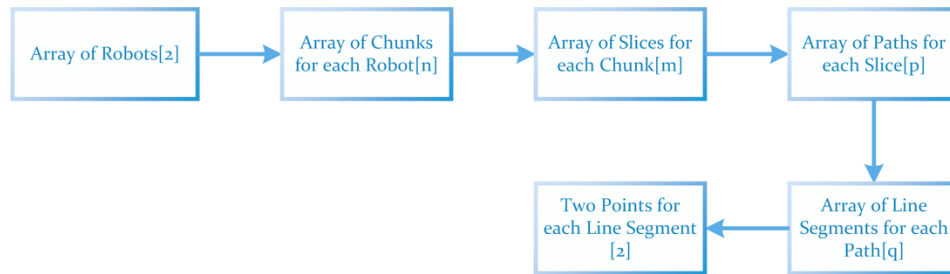


Figure 12. Data structure of the tool paths for the robots

With the entire tool path generated, the slicer needs to generate commands that can be interpreted by the robots to execute the movements. One of the most common type of commands used in a regular 3D printer is G-code commands. In this paper, we use similar commands for our simulator to interpret. For example, we use a “MOVE X Y Z” command, which moves the

robot from its current position to the specified position (X, Y, Z). It is equivalent to a G1 command in G-code and thus make it easy to output the commands to a real robot. Since the tool path is just a list of line segments, this one command will be sufficient to instruct the robot to move along its tool path.

Now that the robots know where to move based on the generated commands, they also need to know when to move. In the situation when one robot's next move is dependent on another robot's printing status, it is necessary to provide a mechanism for the robots to communicate. For example, in our two-robot situation, the second robot must wait until the first robot finishes printing the center chunk to start printing and thus has to know when the first robot finishes printing the center chunk. One direct way is for the robots to have real-time constant communication with each other, but it would significantly increase the complexity when many robots are involved. Luckily, the interdependence of the printing tasks can usually be pre-determined in the chunking or slicing stage. Therefore, we can pre-implant a communication command at the stage when communication is needed. In this paper, we implemented a NOTIFY command, which is inserted behind a MOVE command to notify the other robot that a certain movement has been finished. Because our situation only involves the center-chunk waiting period, only one NOTIFY command is needed for the entire job. The second robot will not begin along its toolpath until it receives the NOTIFY command from the first robot.

In addition to the MOVE and NOTIFY commands, the robot also needs to know when the materials should be deposited. This is because the robot does not print materials along all the tool paths. For example, when the robot is transitioning for one chunk to another, no materials need to be printed. Therefore, we implemented a "TOOL ON/OFF" command to indicate whether the robot should print materials. When "TOOL ON" command is issued, materials will be printed along all the following tool path until a "TOOL OFF" command is issued. Based on the tool path data and how we want to robot to communicate and print materials, we can translate the tool paths into commands for the robots to execute the printing process using the three commands we have implemented. This process is shown in Figure 13.

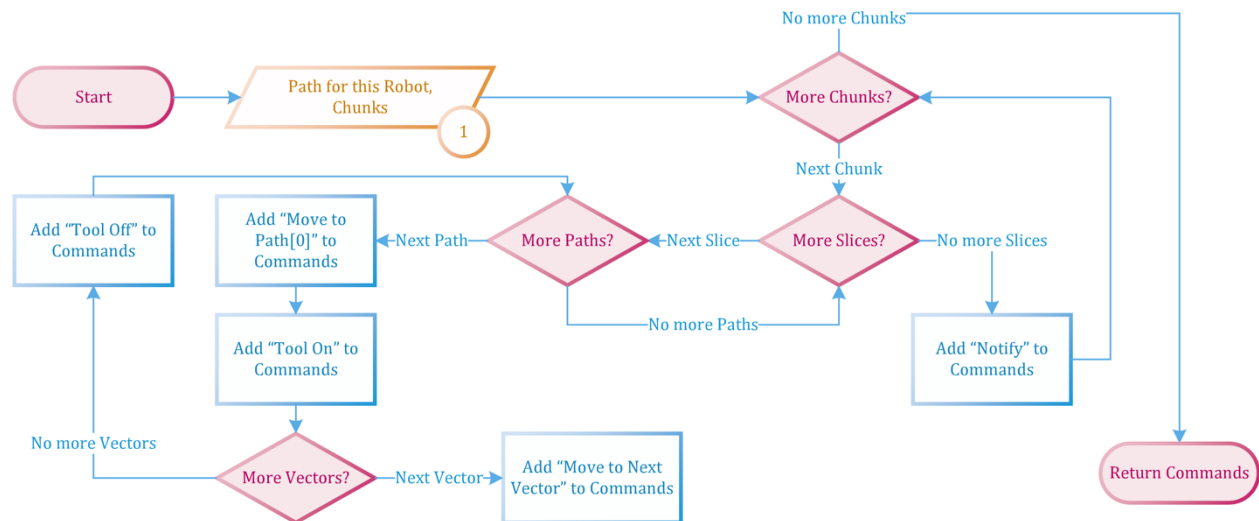


Figure 13. Conversion from the tool paths to robot commands

5. Simulator

The objective of the simulator is to animate the command output from the slicer to visualize the dynamic printing process so that we can evaluate the chunking and slicing results as well as the overall printing strategy. Two essential functions are needed for the simulator: (1) interpret and visualize each single command from the slicer output; (2) animate the sequential commands in correct timing sequence and speed.

5.1 Command Interpretation

The simulator needs to be able to read in the text commands generated from the slicer and parse them and visualize them one by one on the same scene. Since we only have three commands (MOVE, TOOL ON/OFF, and NOTIFY) with well-defined meanings, it is straightforward to interpret them. Figure 14 demonstrates the interpretation of a small set of commands as an example. A cylindrical tube is used to visualize the printed filament.

Sample Commands:

MOVE 0, 0, 0

TOOL ON

MOVE 1, 0, 0

MOVE 1, 1, 0

MOVE 0, 1, 0

TOOL OFF

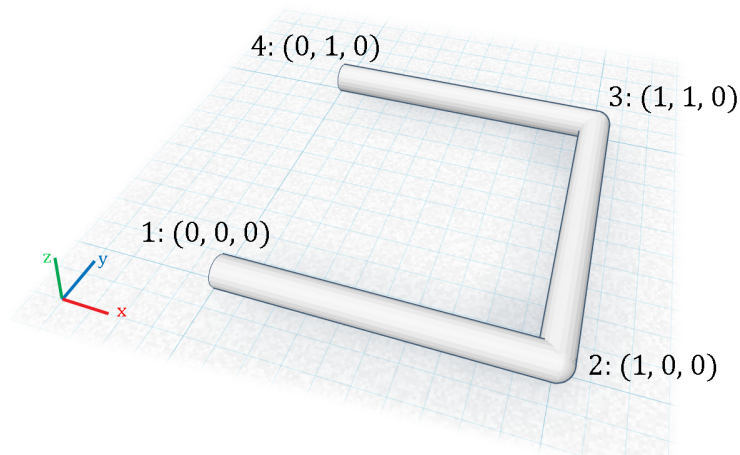


Figure 14. Demonstration of interpretation and visualization of a small set of commands.

5.2 Commands Animation

A simple approach to animate the commands is to visualize each command as one video frame. As we step through each command, we will generate frames for a video. However, this approach does not work well. This is because commands take different times to execute in reality, but they will all appear to take the same time in the animation since each command is represented by one frame in the video. For example, if the robot is moving at the same speed, “MOVE 10, 0, 0” will take 10 times longer than “MOVE 1, 0, 0” in reality, but they will appear to take the same time in the animation.

Therefore, we implement the concept of a “time step”, which is a defined period of time to be simulated between each frame. Based on the robot’s speed and the distance the robot must travel for each command, we can calculate how long the robot should take to execute a given “MOVE” command. To simulate real-time movements, our approach is to execute as many commands in the queue as will fit within the pre-defined time step. If the next command

execution results in going over the time step, then we only execute *part* of that command. This requires a linear interpolation operation to calculate where the robot's extruder will be located along the movement path at the end of the time step.

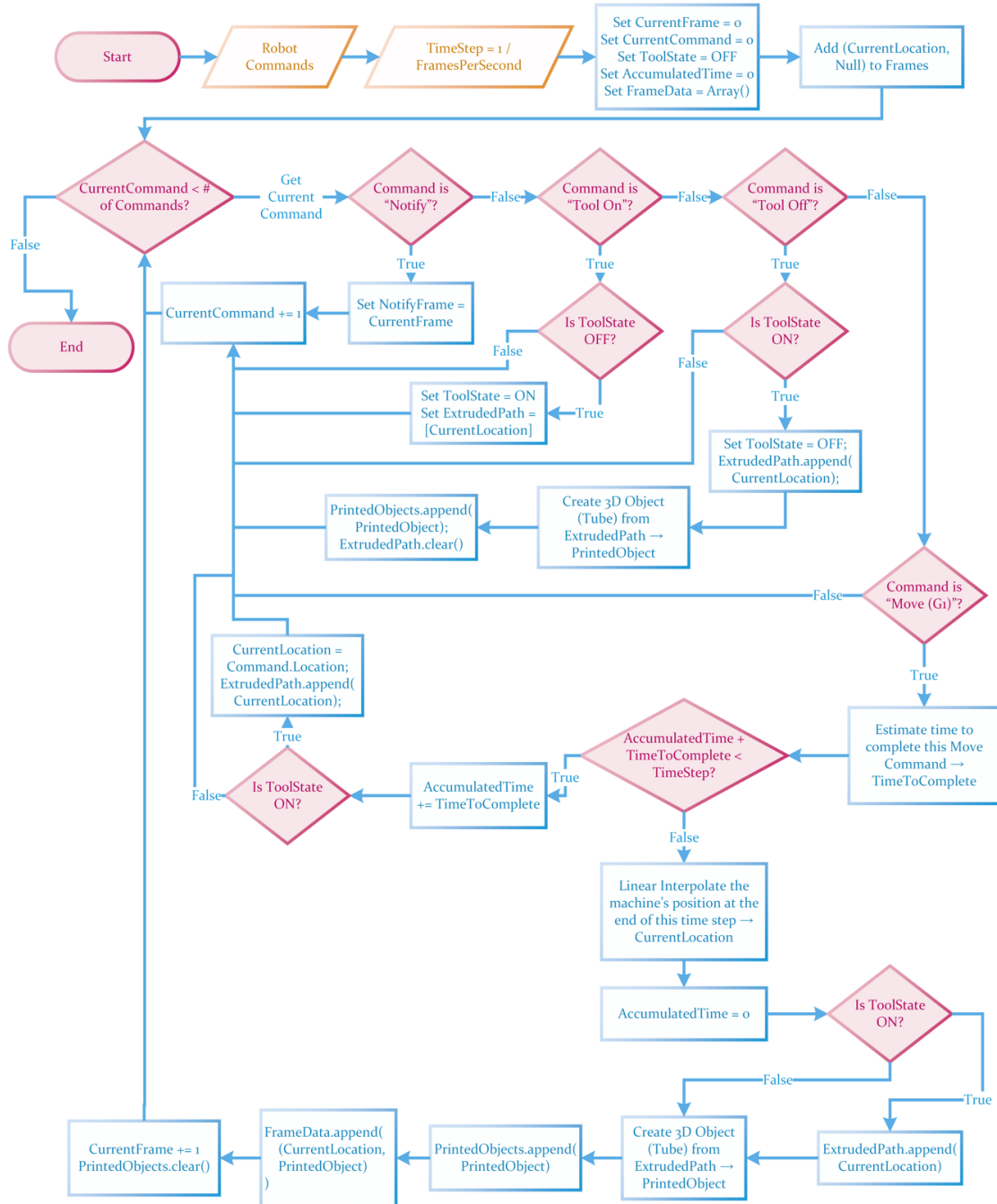


Figure 15. Pseudo code for animating the commands output from the slicer.

In addition to simulating the robot's movements, we have to simulate the extruded material from the printhead. We accomplish this by creating a Bezier curve following the robot's path for every time step, then creating a cylindrical tube (for 3D rendering purposes) that follows the curve. We only create these tubes if the robot's current state is "TOOL ON". Otherwise, no

extruded material is created. Figure 15 includes pseudocode for the path animation function. The function is supplied with a list of commands and the robot's current location and produces a dataset for each frame, which include: (1) the robot's new position; (2) a list of extruded material objects (cylindrical tubes) to display for that frame. The snapshots of the simulated animation are shown in Figure 16.

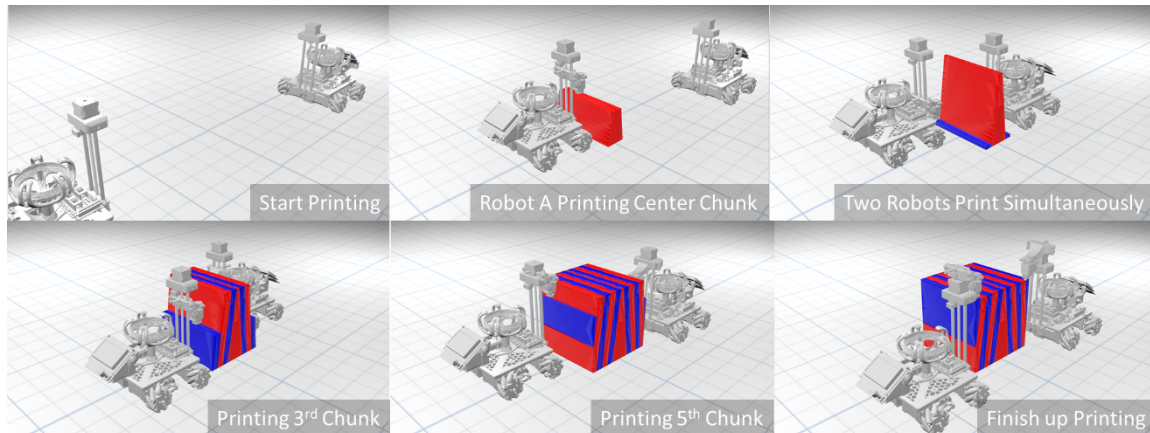


Figure 16. Snapshots of the simulated animation of two robots printing a cube cooperatively.

6. Conclusion

In this paper, we presented a new chunk-by-chunk based approach to 3D printing for a cooperative 3D printing platform. In contrast to a layer-by-layer based approach, this new approach enables a plurality of mobile 3D printers to carry out a printing job simultaneously, which can provide the scalability to 3D printing, in terms of both print size and printing time. The chunk-based strategy also keeps printing localized and thus alleviates large temperature gradient and internal stress that are common to large-scale 3D printing. We first presented a general strategy for the chunk-based printing strategy. A chunker is then developed for two-robot printing to demonstrate a general chunking methodology. A slicer is also developed to slice the chunks and coordinate multiple robots for the printing. A simulator is then developed to visualize and animate the dynamic cooperative 3D printing process to validate the developed chunker and slicer. The results show great promise to a new direction of 3D printing that may provide a path to make 3D printing a mainstream manufacturing technology with autonomous printing robots.

7. Future Work

As the basic viability of chunk-based cooperative 3D printing has been demonstrated in this paper, more work is needed to build this into a technique that rivals current methods of manufacturing. Two mobile printers are not enough to produce the speed required for mass manufacturing, so one of the next steps is to develop scheduling strategies for many robots. In order for cooperative 3D printing to reach its maximum potential, many robots must be able to work in parallel in order to produce parts rapidly. Slicing algorithms for the incorporation of gripper robots that can pick and place pre-manufactured components to be embedded in a 3D printed structure during the 3D printing process can also greatly enhance the capability of 3D printing in fabricating electromechanical devices and other sophisticated products.

8. References

1. ASTM-F42-Committee, *Standard Terminology for Additive Manufacturing Technologies*, in *ASTM International*. 2012: West Conshohocken, PA.
2. A. Ranellucci, *Slic3r: G-code generator for 3D printers*. 2015, Pàgina web oficial del projecte: <http://slic3r.org/about> [últim accés: 25/10/2013].
3. D. Braam, *Cura (software)*, in *Wikipedia*. 2016.
4. K.K.I.S. Slicer). <http://www.kisslicer.com/>. 2016.
5. Skeinforge, <http://fabmetheus.crsndoo.com/overview.php>, in *RepRap Wiki*. 2015.
6. C. Kirschman and C. Jara-Almonte, *A parallel slicing algorithm for solid freeform fabrication processes*. Solid Freeform Fabrication Proceedings, Austin, TX, 1992: p. 26-33.
7. E. Sabourin, S.A. Houser, and J. Helge Bøhn, *Adaptive slicing using stepwise uniform refinement*. Rapid Prototyping Journal, 1996. **2**(4): p. 20-26.
8. S. Lefebvre and L.I.N. Grand-Est. *IceSL: A GPU accelerated CSG modeler and slicer*. in *18th European Forum on Additive Manufacturing (AEFA'13)*. 2013.
9. Repetier, *Repetier-Host*, in *RepRap Wiki*. 2017.
10. N. Koenig and A. Howard. *Design and use paradigms for gazebo, an open-source multi-robot simulator*. in *Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*. 2004. IEEE.
11. T. Bräunl, *The EyeSim Mobile Robot Simulator*. 2000, CITR, The University of Auckland, New Zealand.
12. B. Browning and E. Tryzelaar, *Ubersim: A realistic simulation engine for robot soccer*. Proceedings of Autonomous Agents and Multi-Agent Systems, AAMAS'03, 2003.
13. L. Hugues and N. Bredeche. *Simbad: an autonomous robot simulation package for education and research*. in *International Conference on Simulation of Adaptive Behavior*. 2006. Springer.