A cybermanufacturing framework incorporating deep learning and multi-resolution voxel representations

by

Sambit Ghadai

A dissertation submitted to the graduate faculty
in partial fulfillment of the requirements for the degree of
DOCTOR OF PHILOSOPHY

Co-majors: Mechanical Engineering;
Computer Engineering

Program of Study Committee:
Adarsh Krishnamurthy, Co-major Professor
Soumik Sarkar, Co-major Professor
Baskar Ganapathysubramanian
James Oliver
Chinmay Hegde

The student author, whose presentation of the scholarship herein was approved by the program of study committee, is solely responsible for the content of this dissertation. The Graduate College will ensure this dissertation is globally accessible and will not permit alterations after a degree is conferred.

Iowa State University
Ames, Iowa
2020

Copyright © Sambit Ghadai, 2020. All rights reserved.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Category</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIST OF TABLES</td>
<td>vi</td>
</tr>
<tr>
<td>LIST OF FIGURES</td>
<td>vii</td>
</tr>
<tr>
<td>ACKNOWLEDGMENTS</td>
<td>x</td>
</tr>
<tr>
<td>ABSTRACT</td>
<td>xii</td>
</tr>
<tr>
<td>CHAPTER 1. INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>1.1 Motivation</td>
<td>1</td>
</tr>
<tr>
<td>1.2 Thesis Overview</td>
<td>3</td>
</tr>
<tr>
<td>1.2.1 Intelligent Design for Manufacturing</td>
<td>5</td>
</tr>
<tr>
<td>1.2.2 Process Planning</td>
<td>6</td>
</tr>
<tr>
<td>1.2.3 Product Visualization</td>
<td>6</td>
</tr>
<tr>
<td>1.2.4 Automated Manufacturing</td>
<td>7</td>
</tr>
<tr>
<td>1.2.5 Support Tools</td>
<td>8</td>
</tr>
<tr>
<td>1.3 References</td>
<td>9</td>
</tr>
<tr>
<td>CHAPTER 2. LEARNING LOCALIZED FEATURES IN 3D CAD MODELS FOR MANUFACTURABILITY ANALYSIS OF DRILLED HOLES</td>
<td>10</td>
</tr>
<tr>
<td>2.1 Abstract</td>
<td>10</td>
</tr>
<tr>
<td>2.2 Introduction</td>
<td>11</td>
</tr>
<tr>
<td>2.3 Manufacturability of Drilled Holes</td>
<td>14</td>
</tr>
<tr>
<td>2.3.1 Training Data</td>
<td>16</td>
</tr>
<tr>
<td>2.4 Volumetric Representations for Learning Geometric Features</td>
<td>19</td>
</tr>
<tr>
<td>2.4.1 Voxel Representation of Geometry</td>
<td>19</td>
</tr>
<tr>
<td>2.4.2 Augmenting Volume Representation with Surface Normals</td>
<td>21</td>
</tr>
</tbody>
</table>
2.5 3D-CNN for Learning Localized Geometric Features .......................... 22
  2.5.1 Network Architecture and Hyper-Parameters ........................... 22
2.6 Interpretation of 3D-CNN Output ................................................. 23
2.7 Results and Discussion .................................................................. 24
  2.7.1 Voxelization Timings .............................................................. 25
  2.7.2 Tuning of the Hyper-Parameters .............................................. 25
  2.7.3 Results from Test Data Set ...................................................... 26
  2.7.4 Relationship Between Data Size and Accuracy ......................... 28
  2.7.5 Manufacturability Analysis of a Realistic Part ........................... 31
2.8 Conclusion ................................................................................... 33
2.9 References .................................................................................... 34

CHAPTER 3. MULTI-LEVEL 3D CNN FOR LEARNING MULTI-SCALE SPATIAL
FEATURES .......................................................................................... 37
3.1 Abstract ....................................................................................... 37
3.2 Introduction .................................................................................. 38
3.3 Related Work ............................................................................... 42
  3.3.1 Multi-level Voxel Learning ..................................................... 43
3.4 Multi-level Voxelization ............................................................... 44
3.5 Multi-resolution CNN .................................................................. 47
  3.5.1 Forward Computation of MRCNN ......................................... 48
  3.5.2 Backward Computation of MRCNN ....................................... 50
3.6 Experimental Results & Discussion ............................................. 51
3.7 Conclusion ................................................................................... 55
3.8 References .................................................................................... 55

CHAPTER 4. GPU-ACCELERATED COLLISION FREE NAVIGATION OF VEHICLES IN ENCLOSSED SPACES ......................................................... 60
4.1 Abstract ....................................................................................... 60
4.2 Introduction .................................................................................. 60
4.3 Related Work ............................................................................... 64
4.4 Voxelization ................................................................................ 66
4.5 Collision Detection ..................................................................... 68
  4.5.1 Axis Aligned Culling Region .................................................. 68
  4.5.2 Point-Voxel Collision ............................................................. 69
4.6 Clearance Analysis and Theoretical Guarantees ......................... 70
  4.6.1 Voxel based Minkowski Sums ............................................... 72
# LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table 2.1</td>
<td>Timings and voxel count of the voxelization of the CAD geometries and large-scale model.</td>
<td>26</td>
</tr>
<tr>
<td>Table 2.2</td>
<td>Optimized hyper-parameters with the least validation loss.</td>
<td>28</td>
</tr>
<tr>
<td>Table 2.3</td>
<td>Quantitative performance assessment of the DLDFM on test data sets.</td>
<td>28</td>
</tr>
<tr>
<td>Table 2.4</td>
<td>Illustrative examples of manufacturability prediction and interpretation using the DLDFM framework.</td>
<td>29</td>
</tr>
<tr>
<td>Table 2.5</td>
<td>Comparison of the performance of the DLDFM on the training data.</td>
<td>33</td>
</tr>
<tr>
<td>Table 3.1</td>
<td>Comparisons between different spatial deep learning approaches.</td>
<td>42</td>
</tr>
<tr>
<td>Table 3.2</td>
<td>Comparison of deep learning frameworks with voxel based representation for ModelNet10 object recognition.</td>
<td>53</td>
</tr>
<tr>
<td>Table 5.1</td>
<td>Landscape of adversarial attack strategies on RL agents</td>
<td>86</td>
</tr>
</tbody>
</table>
## LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 1.1</td>
<td>Traditional Manufacturing vs Cyber-manufacturing</td>
<td>3</td>
</tr>
<tr>
<td>Figure 1.2</td>
<td>Cyber-manufacturing Framework</td>
<td>4</td>
</tr>
<tr>
<td>Figure 1.3</td>
<td>Components of Cybermanufacturing framework</td>
<td>5</td>
</tr>
<tr>
<td>Figure 2.1</td>
<td>Framework for deep-learning based design for manufacturability.</td>
<td>12</td>
</tr>
<tr>
<td>Figure 2.2</td>
<td>Different DFM rules based hole examples to classify manufacturable and non-manufacturable geometries</td>
<td>15</td>
</tr>
<tr>
<td>Figure 2.3</td>
<td>Large scale CAD model to test DLDFM network</td>
<td>17</td>
</tr>
<tr>
<td>Figure 2.4</td>
<td>A sample block with a drilled hole with its dimensions highlighted in the projected view</td>
<td>18</td>
</tr>
<tr>
<td>Figure 2.5</td>
<td>Performing voxelization in 2D using GPU rendering</td>
<td>20</td>
</tr>
<tr>
<td>Figure 2.7</td>
<td>Accuracy of DLDFM with different hyperparameters as a function of total number of parameters in each network</td>
<td>27</td>
</tr>
<tr>
<td>Figure 2.8</td>
<td>Comparison of the training performance with and without batch normalization for each layer</td>
<td>27</td>
</tr>
<tr>
<td>Figure 2.9</td>
<td>Hyper-parameter optimization of DLDFM</td>
<td>27</td>
</tr>
<tr>
<td>Figure 2.10</td>
<td>Confusion matrix for the DLDFM performance on test data set.</td>
<td>30</td>
</tr>
<tr>
<td>Figure 2.11</td>
<td>Variation of accuracy with no. of training samples used for</td>
<td>31</td>
</tr>
<tr>
<td>Figure 2.12</td>
<td>Test results on the large scale CAD model using DLDFM framework to detect manufacturable and non-manufacturable drilled holes</td>
<td>32</td>
</tr>
<tr>
<td>Figure 3.1</td>
<td>An illustrative example showing the need for multi scale feature detection in two mechanical parts</td>
<td>40</td>
</tr>
<tr>
<td>Figure 3.3</td>
<td>Multi-level voxelization of B-rep CAD models</td>
<td>44</td>
</tr>
<tr>
<td>Figure 3.4</td>
<td>Data structure for storing the multi-level voxelization.</td>
<td>45</td>
</tr>
<tr>
<td>Figure 3.5</td>
<td>Multi-Resolution Convolutional Neural Network (MRCNN).</td>
<td>47</td>
</tr>
<tr>
<td>Figure 3.6</td>
<td>Mean classification performance with different input resolutions on ModelNet10 dataset</td>
<td>52</td>
</tr>
<tr>
<td>Figure</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
<td></td>
</tr>
<tr>
<td>3.7</td>
<td>GPU memory usage of MRCNN training &amp; equivalent CNN training on specified voxel grid resolutions.</td>
<td></td>
</tr>
<tr>
<td>4.1</td>
<td>Rendering of voxel representation of a Humvee CAD model</td>
<td></td>
</tr>
<tr>
<td>4.2</td>
<td>Point isolation using axis-aligned culling region</td>
<td></td>
</tr>
<tr>
<td>4.3</td>
<td>Example of colliding and non-colliding point-voxels</td>
<td></td>
</tr>
<tr>
<td>4.4</td>
<td>Clearance distance of a boundary voxel of the Humvee voxel model</td>
<td></td>
</tr>
<tr>
<td>4.5</td>
<td>2D example of voxel based Minkowski sum</td>
<td></td>
</tr>
<tr>
<td>4.6</td>
<td>Voxel based minkowski sum convolution cases</td>
<td></td>
</tr>
<tr>
<td>4.7</td>
<td>Game engine rendering of point cloud and vehicle</td>
<td></td>
</tr>
<tr>
<td>4.8</td>
<td>Volumetric rendering of a Humvee CAD model with Minkowski sum operation</td>
<td></td>
</tr>
<tr>
<td>4.9</td>
<td>Game engine renderings of collision if point cloud with voxels</td>
<td></td>
</tr>
<tr>
<td>4.10</td>
<td>Collision detection of voxels in point clouds at two different locations.</td>
<td></td>
</tr>
<tr>
<td>5.1</td>
<td>Visual comparison of MAS and LAS</td>
<td></td>
</tr>
<tr>
<td>5.2</td>
<td>Box plots of PPO Lunar Lander for each attack methods</td>
<td></td>
</tr>
<tr>
<td>5.3</td>
<td>Time vs Attack magnitude along action dimension for LAS attacks</td>
<td></td>
</tr>
<tr>
<td>5.4</td>
<td>Boxplots showing cumulative rewards of PPO agent in Bipedal-Walker</td>
<td></td>
</tr>
<tr>
<td>5.5</td>
<td>DDQN Lunar Lander box plots showing average cumulative reward</td>
<td></td>
</tr>
<tr>
<td>5.6</td>
<td>DDQN Bipedal Walker box plots showing average cumulative reward</td>
<td></td>
</tr>
<tr>
<td>5.7</td>
<td>Comparison of $|\delta_t|$ used across time for a single episode in PPO Lunar Lander</td>
<td></td>
</tr>
<tr>
<td>5.8</td>
<td>$|\delta_t|$ usage plot of DDQN agent in Lunar Lander</td>
<td></td>
</tr>
<tr>
<td>5.9</td>
<td>$|\delta_t|$ usage plot for PPO agent in Bipedal-Walker</td>
<td></td>
</tr>
<tr>
<td>5.10</td>
<td>$|\delta_t|$ usage plot for DDQN agent in Bipedal-Walker</td>
<td></td>
</tr>
<tr>
<td>5.11</td>
<td>Ablation study for PPO Lunar Lander</td>
<td></td>
</tr>
<tr>
<td>5.12</td>
<td>Ablation study for DDQN Lunar Lander</td>
<td></td>
</tr>
<tr>
<td>5.13</td>
<td>Ablation study for PPO Bipedal-Walker</td>
<td></td>
</tr>
<tr>
<td>5.14</td>
<td>Ablation study for DDQN Bipedal-Walker</td>
<td></td>
</tr>
<tr>
<td>5.15</td>
<td>Magnitude of attack with respect to different episodes for Lunar Lander environment with DDQN RL agent</td>
<td></td>
</tr>
<tr>
<td>5.16</td>
<td>Magnitude of attack with respect to different episodes for Bipedal Walker environment with DDQN RL agent</td>
<td></td>
</tr>
<tr>
<td>6.1</td>
<td>Distinction between a regular voxel representation and multilevel voxel representation</td>
<td></td>
</tr>
<tr>
<td>6.2</td>
<td>Outline of direct 3D printing from multi-level voxels</td>
<td></td>
</tr>
<tr>
<td>Figure 6.3</td>
<td>In-depth view of a multi-level voxel grid for direct 3D printing</td>
<td>125</td>
</tr>
<tr>
<td>Figure 6.4</td>
<td>Multi-level Marching Squares algorithm implementation</td>
<td>127</td>
</tr>
<tr>
<td>Figure 6.5</td>
<td>Marching and gap issue while performing MS on multi-level voxel grid</td>
<td>128</td>
</tr>
<tr>
<td>Figure 6.6</td>
<td>Isocontours generated from a layer of Multi-level voxels using MLMS</td>
<td>130</td>
</tr>
<tr>
<td>Figure 6.7</td>
<td>Rendered view of all iso-contours generated from multi-level voxel representation of a scooby CAD model</td>
<td>131</td>
</tr>
<tr>
<td>Figure 6.8</td>
<td>Infill generation using hybrid scan-line in a multi-level voxel grid</td>
<td>133</td>
</tr>
<tr>
<td>Figure 6.9</td>
<td>Isocontour generation and alternating infill patterns in a print layer of scooby model</td>
<td>135</td>
</tr>
<tr>
<td>Figure 6.10</td>
<td>Low and high density infill pattern visualizations in scooby model</td>
<td>136</td>
</tr>
<tr>
<td>Figure 6.11</td>
<td>Layer heights comparison of boundary layer and infill layer</td>
<td>137</td>
</tr>
<tr>
<td>Figure 6.12</td>
<td>3D Print simulation for Scooby model</td>
<td>138</td>
</tr>
<tr>
<td>Figure 6.13</td>
<td>3D Print simulation for Stanford Bunny model</td>
<td>139</td>
</tr>
<tr>
<td>Figure 6.14</td>
<td>3D Print simulation for Turbine model</td>
<td>140</td>
</tr>
<tr>
<td>Figure 6.15</td>
<td>3D Print simulation of heart model from CT-scan image stack</td>
<td>141</td>
</tr>
<tr>
<td>Figure 6.16</td>
<td>Final 3D printed models using multi-level voxels</td>
<td>142</td>
</tr>
</tbody>
</table>
ACKNOWLEDGMENTS

I would first and foremost like to thank my advisors, Prof. Adarsh Krishnamurthy and Prof. Soumik Sarkar for their invaluable guidance and support during my PhD at Iowa State University. I am immensely fortunate to be co-advised by both of them which allowed me to gain a plethora of knowledge and experience essential for conducting research and develop critical thinking. Prof. Krishnamurthy guided me through the beautiful world of Computer Graphics, Solid Modeling and GPU Computing, while Prof. Sarkar introduced me to Deep Learning and its multitude of applications.

I would also like to thank my committee member Prof. Baskar Ganapathysubramanian, Prof. James Oliver and Prof. Chinmay Hegde for their mentoring me in different projects and discussing research ideas. Prof. Oliver’s Virtual Reality course and Prof. Hegde’s Data Analytics course helped me gain important basic knowledge regarding the respective fields and allowed me to implement the theoretical ideas in my research work.

I am grateful to all the members of IDEA Lab and SCS Lab for creating a healthy and collaborative environment that promotes quality research. They gave me the opportunity to disseminate ideas and knowledge in an enjoyable fashion without which my PhD journey would not have been interesting. Specifically, I would like to thank Aditya, Bernard, Kailiang and Anushrut for their contributions in completing the research presented in this thesis. The sound of our discussions will always be the life of Lab of Mechanics. I will always cherish the conversations with Prof. Krishnamurthy and Aditya about anything related and unrelated to research.
I would like to thank my manager at Intel Corporation, Ashwin Muppalla, for giving me the internship opportunity and helping me to learn the essential skills required to work in an industry and conduct applied research at the same time. Many thanks to my teammates at Intel Folsom office for imbibing a teamwork culture in me and guiding me through the internship. I was extremely lucky to use my research skills in a leading technical corporation and experiencing its impact in a larger scale.

I would also like to thank my friends at ISU for their contributions in making this PhD journey one of fun and enjoyment. During these four years, my life outside of research would not have been the same without them. I would like to especially thank my flatmates during the years, first Rahul and Himanshu, and then Amit during the last few months, for making me look forward to the times spent outside of University. My friends played a major role in keeping my spirits high instead of the twist and turn in research for which I will be thankful to all of them, including Aditya, Akshay Nene, Akshay Kore, Abhijit, Alka, Amit, Aseem, Ashish, Bernard, Dinesh, Himanshu, Kailiang, Mihir, Rahul, Reetam, Sahiti, Sarthak, Shubham Mutha, Shubham Khoje, Yash, and many more. I would like to thank my friends from undergraduate studies at NIT Rourkela, specifically Abhishek, Anubhav Patra, Anubhav Abhinav, Arpita, Ipsit, Rajdeep, Santak, Shweta, and others, for always being there for me.

Special thanks to my family members for their unrelenting support during my doctorate studies and allowing me to pursue my higher studies in a far off country. I am grateful to my father and mother for always believing in me and my decisions regarding my career. I would like to thank my sister for always being there for my parents and me through all the highs and lows during my PhD work.

Last but not the least, I want to thank the Supreme God for making this journey one of the biggest life experiences for me till now and giving me the courage to face all adversities and achieve success.
ABSTRACT

Cybermanufacturing (CM) is a modern concept involving predictive analytic operations and information technology to aid the manufacturing industry in better decision making for design and manufacturing processes. This thesis presents a data-driven intelligent Cybermanufacturing framework for the effortless design and manufacturing of a product.

While traditional manufacturing systems are iterative and especially require skilled operators in the process, CM systems alleviate this issue by making intelligent predictions without specialists’ involvement. CM systems operate with a network and data-rich environment involving interaction within and between virtual and physical spaces resulting in an effective decision support system. The broad objective of this research is to define and establish a framework of a cyber-physical system consisting of such virtual and physical systems to confront various departments of a manufacturing process.

The first stage in most of the iterative manufacturing processes is a product design that is compliant with certain design specifications and requirements. However, this is not a one-stop solution; to realize the final design, a product goes through multiple iterations between design and manufacturing stages to be compliant with the existing manufacturing paradigm. To tackle this issue, we have developed data-driven decision support for an intelligent design for manufacturing (DFM) framework using a volumetric representation (voxels) of 3D CAD models and deep neural networks to make high-quality predictions of the manufacturability of a part or product without requiring domain expertise of the user. We have developed a manufacturing process planning framework that
detects such features irrespective of its size by hierarchically representing 3D CAD models as volumes on multiple scale levels (multi-level voxels) and facilitating scale-variant feature learning through the implementation of a multi-level Deep Neural Network to make decisions from hierarchical data.

Along with virtual decision support systems for design and manufacturing, CM systems also involve actual manufacturing in the physical space using machines and robotic environments. We have developed an automated manufacturing module that includes an algorithm for direct 3D printing from voxels and optimization based robust reinforcement algorithm.
CHAPTER 1. INTRODUCTION

1.1 Motivation

Traditional design and manufacturing systems have been transforming the manufacturing industry since the industrial revolution in a way to produce high quantity and cheap products. However, to produce high-quality products, the process gets quite expensive, because it involves re-iteration of the process and requires design and manufacturing expertise. Cybermanufacturing (CM) is a cyber-physical system (CPS) framework that alleviates these issues by providing an intelligent decision support system and efficient network management strategies of various virtual applications and machines for design, manufacturing, and production. Current studies manufacturing systems promote CM as the primary driving force behind Industry 4.0 [1, 2, 5], which is being adopted by countries with large industrial sectors to drive overall economic success. Transformation to Industry 4.0 however, requires smooth digitalization of the manufacturing sector, which is mostly governed by CM. Fundamental impacts of Industry 4.0, such as interoperability, decentralization of information, heightened flexibility, and real-time data flow, require advanced functionality such as smart connection, data-to-information conversion, digital twin, and cognition and configuration.

In the current data-rich world, CM facilitates the use of interconnected systems and data from multiple source systems to perform predictive and prescriptive operations [3, 4, 6]. There exist many strategies governing a CM process consisting of various soft-
ware and hardware tools, data-driven methods for predictive analysis augmenting the decision support system, Internet-of-Things (IoT) based network systems, communications protocols among software and machine components and other such CPS methodologies. By combining all these individual strategies for a manufacturing ecosystem, CM enhances traditional manufacturing in terms of time, cost, quality, compliance, and security. The primary functional difference between traditional manufacturing practices and CM is shown schematically in Figure 1.1.

Integrating the individual components with efficient inter-operability and interaction is currently the need to create an advanced manufacturing system capable of contributing to Industry 4.0 through CM. Specifically, in the design and manufacturing domain, intelligent systems are essential to effectuate effortless and efficient decision making. The huge amount of data being generated from each of the components of a manufacturing system needs to be converted to useful information for this purpose, which is infeasible for a human due to the involvement of complex systems. In addition, maintaining the reliability and security of the process is equally important. With the involvement of IoT, sensor networks, and big data, CM processes have the potential to get increasingly complex in a short amount of time. Hence, an efficient framework for CM to overcome these issues by minimizing human intervention while contributing to the decision making process with an intelligent decision support system for advanced manufacturing is the need of the hour.

This thesis describes a novel framework for a CM process that integrates manufacturing systems with Industry 4.0. We have developed an intelligent decision-support system for design and manufacturing augmented by a data-driven framework for process planning. We also devised advanced strategies for system security of machine automation frameworks, developed a novel additive manufacturing paradigm for quality consistency, and devised a visualization method to support the part design process. Fur-
Figure 1.1: Traditional Manufacturing vs Cyber-manufacturing

ther, we maintain data interoperability between each of the individual components of the framework to facilitate straightforward integration with the help of voxels.

1.2 Thesis Overview

The core components of the CM framework presented in this thesis are as follows:

- **Voxel Operations**: Voxels or 3D pixels are used to represent a 3D CAD model as a volumetric grid that stores the occupancy information of the model. Capturing the volume occupancy allows voxels to be used in various applications in advanced design, manufacturing, visualizations, and analysis []. However, to capture all the fine details of a CAD model, the voxel grid’s resolution needs to be very high, thus inducing high memory and computation requirements. Several hierarchical voxel representation schemes have been recently developed to alleviate this issue, such as octrees, kD-trees, and multi-level voxels. While octrees and kD-trees have been mostly used in prior works, they have limited usage in many applications due to their unstructured characteristics. In this thesis, we facilitate the use of multi-level
voxels and develop CM workflows that efficiently uses its sparse yet structured properties for various applications.

The complete CM framework presented in this thesis is driven by 3D CAD models represented using voxels, especially multi-level voxels and all the virtual applications are developed accordingly. This creates a consistent and uniform data flow in the CM process, while volumetric information enables better predictions with Deep Learning algorithms. In addition, we are able to develop highly parallel GPU based algorithms using voxels due to the structured volumetric grids.

- **Deep Learning for Computer Vision**: Deep learning-based paradigms are employed for efficient decision making from a huge amount of data. For 3D computer vision applications, Deep Learning allows the better generalization of the predictions through the use of Neural Networks, specifically Convolutional Neural Networks (CNN), to learn from 3D CAD voxel models. Specifically, we use 3DCNNs to exploit the volumetric representation of CAD models that provides feature-rich data for better prediction capabilities.
The primary modules discussed in the thesis concerning the novel CM process are shown in Figure 1.2. Each of the components of the CM framework are shown in Figure 1.3 and described in the following subsections.

1.2.1 Intelligent Design for Manufacturing

Design for Manufacturing (DFM) in a manufacturing process plays an important role in determining the feasibility of manufacturing a product based on its design. This is an iterative process in traditional manufacturing scenarios where design and manufacturing engineers go over a design to make it feasible for manufacturing based on the features of the design and the machines’ availability. This module, described in Chapter 2 in the thesis, explains a voxel-based design for manufacturing framework to determine
the manufacturability of drilled holed using deep learning. It explores the capability of Convolutional Neural Networks to learn features of voxel 3D models based on certain rules that are used as priors in the process.

1.2.2 Process Planning

The process planning step in a manufacturing framework defines the sequence of manufacturing operations a stock part needs to realize the final physical product. This depends on the number and type of machines available for the manufacturing and the expertise of the individuals performing this task. In the CM process described in the thesis, the process planning step, detailed in Chapter 3 is driven by object and feature recognition of the 3D part design that determines the manufacturing processes to be employed. The feature recognition task is performed with the use of Multi-level voxel representation of 3D CAD models that attenuates the volume representation at different scales. To effectively learn the features from this hybrid data representation, a novel multi-level 3DCNN is developed, which is a predictive data-driven hierarchical learning paradigm.

1.2.3 Product Visualization

In the modern CM systems, visualizing the design stages and manufacturing steps to be followed is an integral area. With the advent of visualization techniques such as Virtual Reality (VR) and Ray Tracing (RT), the extent to which a part or design can be explored is still a growing research focus. We have developed a visualization and model path planning module that deals with point clouds, B-Rep models, and voxel models to effectively visualize and interact with a 3D CAD model with an enhanced VR functionality. This framework, as explained in Chapter 4 is used to develop a Ship Dimensional Analysis Tool (SDAT) for the United States Marine Corps that allows them to make design changes to a new or existing vehicle model/design (ground, air, and amphibious
vehicles) to productively use the existing space in Naval ships for maneuvering and storing using collision detection. This tool also ties up to the process planning stage of the CM framework if a manufactured part is to be moved from one machine to another in an optimal path of least resistance without hindering the quality of the produced part.

1.2.4 Automated Manufacturing

In CM systems, physical manufacturing is manifested as a component of the network comprising the virtual application space and the physical space. Each of the virtual space modules that drive the decision support system for cybermanufacturing augments the final manufacturing protocol to be followed in the physical space to be followed by a group of machines and automated robots. For the least human intervention use of robots and other such automated components are crucial. Hence, this thesis describes an algorithm to monitor the health of such automated robots used in manufacturing. With the advent of the Internet of Things and big data, employing robots with self-learning capability using techniques such as Reinforcement Learning (RL) and Deep Learning ensures the generalization and better decision making capability of such robots. We have developed a robust reinforcement learning framework, analyzed in Chapter 5, that generates intelligent and optimized adversarial attack strategies that can be used to realize a CPS’ vulnerabilities. These strategies can be used to diagnose adversarial attacks on an RL-driven robot that affects its actuators in the RL action space. This eventually leads to a reduction in manufacturing process cost and product validity by diagnosing and prevent unwanted adversarial attacks on the CM process.

We have also developed a Direct 3D Printing algorithm that exploits the data representation (voxels) used in the presented CM framework to perform additive manufacturing operations and produce high-quality parts. This framework uses multi-level voxels to accurately 3D print a high fidelity model, which reduces the staircase effect generally as-
associated with voxels. We have developed a variable height printing strategy to tackle the manufacturing of the surface and inside of the model separately from multi-level voxels. An extra benefit of directly printing from voxels is that this approach can be used to manufacture 3D prints from CT scan and MRI data. This algorithm keeps the data homogeneity intact in the end-to-end CM framework.

1.2.5 Support Tools

Various support tools were also developed that enhance and add to the functionality of each of the CM process components discussed in this thesis. These include:

- Standard and multi-resolution voxelization of 3D CAD models
- Ray-marching algorithm for fast voxelization of 3D CAD models
- Ray-marching methods on voxel grids for virtual microstructure testing
- GPU-accelerated voxel-based Minkowski sums to control the collision accuracy in a point cloud based enclosed spaces
- Multi-level marching squares algorithm to extract accurate iso-contours from multi-level voxel representation of CAD models and medical imaging data
- Hybrid scan-line algorithm to enable sparse and variable height infill structures from multi-level voxels for additive manufacturing
- Voxel-based fast distance transforms to expand the volume occupancy information of voxels
- Additive manufacturing as-manufactured model generation
- Virtual Reality based interactive visualization of voxels using game-engine
This thesis is arranged as follows. Chapters 2, 3 and 4 deal with the cyber space aspect of CM. We develop a Design for Manufacturing framework using 3D convolutional neural network on voxel representations of CAD models in Chapter 2. In Chapter 3 we describe a object recognition framework for process planning to detect multi-scale features using multi-level 3D CNN. We present a collision free navigation framework of vehicles in point cloud based enclosed spaces in Chapter 4. Chapter 5 and 6 describe the automated manufacturing components of the physical space in the CM framework. In Chapter 5 we develop optimization based attack strategies on the action space of deep reinforcement learning agents. Finally in Chapter 6 we describe a direct 3D printing method to print from multi-level voxel models and medical imaging data.

1.3 References


CHAPTER 2. LEARNING LOCALIZED FEATURES IN 3D CAD MODELS FOR MANUFACTURABILITY ANALYSIS OF DRILLED HOLES

A paper published in Computer Aided Geometric Design
Sambit Ghadai, Aditya Balu, Adarsh Krishnamurthy and Soumik Sarkar

2.1 Abstract

We present a novel feature identification framework to recognize difficult-to-manufacture drilled holes in a complex CAD geometry using deep learning. Deep learning algorithms have been successfully used in object recognition, video analytics, image segmentation, etc. Specifically, 3D Convolutional Neural Networks (3D-CNNs) have been used for object recognition from 3D voxel data based on the external shape of an object. On the other hand, manufacturability of a component may depend on both its external shape and the local features. Learning these local features from a boundary representation (B-rep) CAD model is challenging due to lack of volumetric information. In this paper, we learn local features from a voxelized representation of a CAD model and classify its manufacturability. Further, to enable effective learning of localized features, we augment the voxel data with surface normals of the object boundary. We train a 3D-CNN with this augmented data to identify local features and classify the manufacturability. However, this classification does not provide information about the source of non-manufacturability in a
complex component. Therefore, we have developed a 3D-CNN based Gradient-weighted Class Activation Mapping (3D-GradCAM) method that can provide visual explanations of the local geometric features of interest within an object. Using 3D-GradCAM, our framework can identify difficult-to-manufacture features, which allows a designer to modify the component based on its manufacturability and thus improve the design process. We extend this framework to identify difficult-to-manufacture features in a realistic CAD model with multiple drilled holes, which can ultimately enable development of a real-time manufacturability decision support system.

2.2 Introduction

Deep learning (DL) algorithms, designed to hierarchically learn multiple levels of abstractions from data, have been extensively used in computer vision [8, 9, 13]. Specifically, 3D-Convolutional Neural Networks (3D-CNN) have been used extensively for object recognition, point cloud labeling, video analytics, human gesture recognition, object shape retrieval etc. [6, 10–12, 18, 20]. Object recognition [15] has been one of the most challenging problems in the area of computer vision and pattern recognition. Early DL-based approaches used simple projection of the 3-dimensional object to a 2-dimensional representation such as depth images or multiple views to recognize an object. However, there is significant loss of geometric information while using a 2D representation of a 3D object. Therefore, it is difficult to learn about the local or internal features of the object using these projection-based techniques. In this paper, we make use of voxelized 3D representation of the object, augmented with surface normal information, to identify localized features in an object.

Voxelized models have been successfully used for object recognition [10, 11, 18]. Generally, point cloud data is converted to a volume occupancy grid or voxels to represent the
model and identify the object. However, voxel-based occupancy grid representation does not inherently have information regarding the surfaces of the object without additional processing. It is also not robust enough to capture information about the location, size, or shape of a feature within an object. Providing additional information about the geometry increases the accuracy of object detection. In this paper, we propose to use normal information of the surfaces of the object, in addition to the volume occupancy information, to efficiently learn the local features. Learning local features in a geometry is different from object recognition. Object recognition is a classification problem, where the object is classified based on a collection of features identified in the object. In this work, we make use of a DL network to learn local feature descriptors without any specific input to describe the features. Based on the feature descriptors that are learned, we make use of a semi-supervised methodology to classify the part based on the local features of interest.
One of the applications of the aforementioned methodology explored in this paper, is to identify difficult-to-manufacture features in a CAD geometry and ultimately classify its manufacturability. A successful part or product needs to meet its specifications, while being feasible to manufacture. In general, manufacturability feedback is provided to the designer only after the design has been finalized. This leads to an iterative process, often leading to longer product development times and higher costs. There are different handcrafted design for manufacturability (DFM) rules that ensure manufacturability of a design. For this purpose, the hierarchical architecture of DL can be used to learn increasingly complex features by capturing localized geometric features and feature-of-features. Thus, a deep-learning-based design for manufacturing (DLDFM) tool can be used to learn the various DFM rules from different examples of manufacturable and non-manufacturable components without explicit handcrafting.

A primary concern while examining the manufacturability of CAD geometries using a DL based approach is the black-box nature of such deep networks. Interpretation of the decision making process in the form of visual explanations is essential for extracting the local features in an object that effectuates its non-manufacturability. Visual detection of local features further enables re-designing of the component to be manufacturable. Recent major work on interpreting DL output by Selvaraju et al. [14] makes use of a 2D gradient-weighted class activation map for producing visual explanations of the CNN’s decision making process for object recognition in images. In this work, we extend the GradCAM to 3D objects for interpretation of 3D-CNN’s outputs and visualizing the regions that give rise to non-manufacturability conditions in the objects.

In this paper, we present a 3D Convolutional Neural Network (3D-CNN) (shown in Figure 2.1) based framework that will learn and identify localized geometric features from an expert database in a semi-supervised manner. Further, this framework is trained in the context of manufacturability with different CAD models classified as manufacturable and
non-manufacturable based on traditional DFM rules. The main contributions of this paper include:

- GPU-accelerated methods for converting CAD models to volume representations (voxelization augmented with surface normals), which can be used to learn localized geometric features.
- Application of deep learning to manufacturability analysis of drilled holes.
- A novel voxel-wise 3D gradient-weighted feature localization based on the 3D-CNN framework to identify local features that are non-manufacturable.

This paper is arranged as follows. In Section 2.3, we discuss the design for manufacturability (DFM) rules used for generation of the datasets for training and testing the 3D-CNN. We explain about the volume representations that we use for 3D-CNN in Section 2.4. In Section 2.5, we discuss the details of 3D-CNN, including the network architecture and the hyper-parameters. In Section 2.6, we explain 3D gradient weighted class activation mapping for identifying the localized geometric features. Finally, in Section 3.6, we show the results of the deep learning based design for manufacturability (DLDFM) framework in classifying manufacturable and non-manufacturable CAD models and capability of the model to identify localized geometric features in a large realistic CAD model.

2.3 Manufacturability of Drilled Holes

Design for Manufacturing (DFM) rules for drilling have been traditionally developed based on the parameters of cylindrical geometry and geometry of the raw material. In this paper, we show an application of our approach to learn localized geometric features to identify non-manufacturability of drilled holes. Deciding the manufacturability of a part
Figure 2.2: Different DFM rules based hole examples to classify manufacturable and non-manufacturable geometries.

is framed as a binary classification problem. We apply 3D-CNN to learn the parameters to classify for manufacturability.

The important geometric parameters of a hole are the diameter, the depth, and position of the hole. However, there are certain additional geometric parameters that do not contribute to manufacturability analysis, but might affect the machine learning framework. For example, the face of the stock on which the hole is to be drilled does not affect the manufacturability of the hole, but need to be considered while training because the volumetric representation of the CAD model is not rotationally invariant.

In our DLDFM framework, the following DFM rules are used to classify the drilled hole as manufacturable as shown in Figure 2.2:

1. **Depth-to-diameter ratio**: The depth-to-diameter ratio should be less than 5.0 for the machinability of the hole [2, 3]. It should be noted that this rule is generic and applicable for all materials.

2. **Through holes**: Since a through hole can be drilled from both directions, the depth-to-diameter ratio for a through hole should be less than 10.0 to be manufacturable.
3. **Holes close to the edges**: A manufacturable hole should be surrounded with material of thickness at least equal to the half the diameter of the hole.

4. **Thin sections in the depth direction of the hole**: A manufacturable hole should should have material greater than half the diameter along the depth direction.

The preceding rules are used to generate the ground truth manufacturability data for the training set, which is then used to learn the manufacturable and non-manufacturable features by the DLDFM Framework. However, it should be noted that for a DLDFM framework, one need not explicitly mention the rule. Rather, for industrial applications, one can train the DLDFM framework using the industry relevant historical data available in the organization, which need not be strictly rule based. The historical data can also be based on experience during previous attempts to manufacture a part. Thus, this DLDFM framework is an attempt to generate a DFM framework based on few basic local features. For more complicated shapes, one can augment the training set and then re-train the DLDFM framework, which can then be used for analyzing complex parts for manufacturability. This eliminates explicit hand-crafting of rules, ultimately leading to a better manufacturability analysis.

### 2.3.1 Training Data

Based on the DFM rules for drilling, different synthetic sample solid models are generated using a CAD modeling kernel. A large component can be considered to be an union of these synthetic blocks as shown in Figure 2.3. We use ACIS [17], a commercial CAD modeling kernel to create the solid models. Cuboids and Cylinders of different sizes that fit in a bounding box of 5.0 inches with different sizes of drilled holes were created (Figure 2.4) as the training set. The diameter of the holes is randomly chosen from 0.1in. to 1.0in. with a discrete resolution of 0.1in. Similarly, the depth of the hole is randomly
Figure 2.3: Large scale CAD model to test DLDFM network. The holes in red are non-manufacturable and the green are manufacturable based on the DFM rules as discussed in Section 2.3. Orange colored hole is a marginal case of manufacturability. The two yellow holes are manufacturable individually but are non-manufacturable due to their proximity to each other.

chosen from 0.5in. to 5.0in. with an discrete resolution of 0.1 in. The holes are generated at different positions on the face of the cube by varying the value of PosX and PosY randomly with a discrete resolution of 0.1in (Figure 2.4). In addition, the holes are generated randomly on any of the six faces. Repetition of the models is not allowed and thus, all the models generated are completely different from other. A total of 800,000(approx.) possible models can be generated. Taking into advantage of the fact that 3D-CNN can learn the features and thus would not need much data about the variation of these models, we generate only a subset of the models. After the CAD models are generated using the solid modeling kernel, they are classified for manufacturability using the DFM rules for drilled holes.

After the B-Rep models are generated using the CAD modeling kernel, they are converted to volume representations. One of the framework design choices is to choose an appropriate voxel grid for the model. A fine voxelization of the model, while capturing all the features accurately, might be computationally expensive for training the 3D-CNN. Even a voxelization resolution of $64 \times 64 \times 64$ pushes the limits of the GPU and CPU
memory during the training, and hence, the parameters of the 3D-CNN have to be tuned for optimal performance. The complete data is split into training set and validation set.

In order to test the performance of the DLDFM network, a new set of CAD models were generated. The CAD geometry generated in this set is different from the CAD models used in training the DLDFM. The models generated in test-set do not have the same depth or diameter values as those in the models of training set. For the test data, the following parameters are varied to generate the samples:

- Diameter values from 1.1 to 1.5 are used for the representative test data. Diameter values from 0.1 to 1.0 inches were used for training.
- Position of the holes that are not used while training.

In addition to that we generate a few more models for testing that does not resemble training data. These models have geometries with the same primary hole parameters (depth, diameter, and position of the hole), but having additional or different external features. The details of the geometries in the non-representative data set is given below.

**Multiple holes:** The DLDFM has been trained to analyze the manufacturability of a single drilled hole. However, in an a designed component, the features may not be in-
dependent; there can be multiple features, each of which may or may not be manufacturable. Moreover, it is possible that each of the features themselves are manufacturable, but due to their proximity or interaction with other features, the part may become non-manufacturable. Hence, we test the ability of the DLDFM framework to analyze the manufacturability of a part with two holes.

**L-shaped blocks:** All the models in the training set have an external cubical or cylindrical shape. Hence, to test the capability of DLDFM to capture the manufacturability of a hole irrespective of the external geometry, we use L-shaped Block and cylinders with holes. The rules established in Section 2.3 also apply to this geometry.

### 2.4 Volumetric Representations for Learning Geometric Features

Traditional CAD systems use boundary representations (B-Reps) to define and represent the CAD model [7]. In B-Reps, the geometry is defined using a set of faces that form the boundary of the solid object. B-Reps are ideally suited for displaying the CAD model by first tessellating the surfaces into triangles and using the GPU to render them. However, learning spatial features using B-Rep can be challenging, since the B-Rep does not contain any volumetric information.

#### 2.4.1 Voxel Representation of Geometry

The use of voxelized shape representation allows a digital representation of the CAD model, where each voxel of the model can be represented using a binary digit corresponding to the voxel being inside or outside the model. Using this method, the entire model can be represented using a long string of binary digits that can be used as input for training the machine learning network.
In our framework, we convert the B-Rep CAD model to a volumetric occupancy grid of voxels. However, voxelizing a B-Rep CAD model is a compute intensive operation, since the center of each voxel has to be classified as belonging to either inside or outside the model. In addition, thousands of models need to be voxelized during training. Traditional CPU voxelization algorithms are too computationally slow for training the machine learning network in a reasonable time frame. Hence, we have developed methods for accelerated voxelization of CAD models using the graphics processing unit (GPU). These GPU methods are more than 100x faster than the existing state-of-the-art CPU-based methods and can create a voxelized representation of the CAD model with more than 1,000,000,000 voxels. Having a high resolution voxelization will enable us to capture small features in a complex CAD model.

To create the voxelized CAD model, we construct a grid of voxels in the region occupied by the object. We then make use of a rendering-based approach to classify the voxel centers as being inside or outside the object. A 2D example of the method is shown in Figure 2.5; the method directly extends to 3D. The CAD model is rendered slice-by-slice
by clipping it while rendering. Each pixel of this clipped model is then used to classify the voxel corresponding to the slice as being inside or outside the CAD model. This is performed by counting the number of fragments that were rendered in each pixel using the stencil buffer on the GPU. After the clipped model has been rendered, an odd value in the stencil buffer indicates that the voxel on the particular slice is inside the CAD model, and vice versa (Figure 2.5). The process is then repeated by clipping the model with a plane that is offset by the voxel size. Once all the slices have been classified, we get the complete voxelized representation of the CAD model.

The time taken to perform the classification is the sum of the time taken to tessellate the model once and the total time taken to render each slice. As an example, the total time taken to voxelize the hole block is 0.133 seconds. These timings are obtained by running our voxelization algorithm on a Intel Xeon CPU with 2.4 GHz processor, 64 GB RAM, and an NVIDIA Titan X GPU.

2.4.2 Augmenting Volume Representation with Surface Normals

While a digital voxelized representation may be sufficient for a regular object recognition problem, it may not be enough to capture the detailed geometric information required for identifying local features of interest within the object. In particular, information about the boundary is lost in a voxel grid and the local region around a voxel grid need to be analyzed to identify a boundary voxel. To overcome this challenge in voxelized representations, we augment the voxel occupancy grid with the surface normals of the B-Rep geometry. First, we identify the boundary voxels of the B-Rep geometry. We consider each voxel as an axis-aligned bounding-box (AABB) using the center location and size of each voxel. We then find all the triangles of the B-Rep model that intersect with the AABBs. Finally, we average the surface normals of all triangles that intersect with
each AABB. The $x, y, \& z$ components of the surface normals are then embedded in the voxelization along with the occupancy grid.

2.5 3D-CNN for Learning Localized Geometric Features

The voxel based representation with the surface normals of the CAD model can be used to train a 3D-CNN that can identify local features. 3D-CNNs have mostly been used for complete 3D object recognition and object generation problems. However, to the best of the authors’ knowledge, this is the first application of 3D-CNNs on identifying local features with applications in manufacturability analysis.

2.5.1 Network Architecture and Hyper-Parameters

The input to the 3D-CNN is a voxelized CAD model. The input volumetric data is first padded with zeros before convolution is performed. Zero padding is necessary in this case to ensure that the information about the boundary of the CAD model is not lost while performing the convolution. The convolution layer with RELU activation is followed by a batch normalization layer and a Max. Pooling layer. The same sequence of Convolution, batch normalization and Max. Pooling is again used. A fully connected layer is used before the final output layer with sigmoid activation. The hyper-parameters of the 3D-CNN that needs to be tuned in order to ensure optimal learning. The specific hyper-parameters used in our framework are listed in Section 3.6.

The model parameters $\theta$, comprised of weights $W$ and biases, $b$ are optimized by error back-propagation with binary cross-entropy as the loss function [5] using the ADADELTA optimizer [19]. Specifically, the loss function $\ell$ to be minimized is:

$$\ell = -y \log \hat{y} - (1 - y) \log(1 - \hat{y})$$ (2.1)
where \( y \in \{0, 1\} \) is the true class label and \( \hat{y} \in \{0, 1\} \) is the class prediction. For training the network, the CAD models that were generated based on the method illustrated in section 2.3 were used.

### 2.6 Interpretation of 3D-CNN Output

The trained 3D-CNN network can be used to classify the manufacturability of any new geometry and can be treated as a black-box. However, interpretability and explainability of the output provided by the 3D-CNN is essential to understand the features learned by the model. In this paper, we attempt to visualize the input features that lead to a particular output and if possible modify it. A similar approach was used in object recognition in images by using class activation maps to obtain class specific feature maps [14]. The class specific feature maps are obtained by taking a class discriminative gradient of the prediction with respect to the feature map to get the class activation. In this paper, we present the first application of gradient weighted class activation map (3D-GradCAM) for 3D object recognition.

In order to get the feature localization map using 3D-GradCAM, we need to compute the spatial importance of each feature map \( A_l \) in the last convolutional layer of the 3D-CNN, for a particular class, \( c \) (\( c \) can be either non-manufacturability or manufacturability, for the sake of generality) in the classification problem. This spatial importance for each feature map can be interpreted as weights for each feature map; it can be computed as the global average pooling of the gradients back from the specific class of interest as shown in Eqn. 2.3.

The cumulative spatial activations that contribute to the class discriminative localization map, \( L_{3DGradCAM} \), is computed using

\[
L_{3DGradCAM} = ReLU \left( \sum_l a_l \times A^l \right),
\]

(2.2)
where $\alpha_l$ are the weights computed using

$$
\alpha_l = \frac{1}{Z} \times \sum_i \sum_j \sum_k \frac{\partial y^c}{\partial A_{ijk}}.
$$

(2.3)

We can compute the activations obtained for the input part using $L_{3DGradCAM}$ to analyze the source of output. The heat map of $(L_{3DGradCAM})$ is resampled using linear interpolation to match the input size, and then overlaid in 3D with the input to be able to spatially identify the source of non-manufacturability. This composite data is finally rendered using a volume renderer.

We make use of a GPU-based ray-marching approach to render this data. The rendering is parallelized on the GPU with each ray corresponding to the screen pixel being cast independently. The intersection of the ray with the bounding-cube of the volumetric data is computed, and then the 3D volumetric data is sampled at periodic intervals. The sum of all the sampled values along the ray is then computed, converted to RGB using a suitable color-bar, and rendered on the screen. Table 2.4 shows different volumetric renderings of the composite 3D-GradCAM data.

### 2.7 Results and Discussion

The different CAD geometries generated as explained in Section 2.3.1 are classified to be manufacturable or non-manufacturable based on the rules discussed in Section 2.3. The B-Rep CAD geometries are converted to volumetric representation using the voxelization method explained in the Section 2.4. The grid size of $64 \times 64 \times 64$ is used for the volumetric representation in order to represent the geometry with sufficient resolution. We first trained the DLDFM network using only the voxelized representation of the CAD geometry. We also trained another network to use the surface normal information (i.e. $x, y, z$ components) in addition to the voxelized representation of the CAD geometry.
These are considered as four channels of the 3D-CNN input to train the second DLDFM network.

We generated 9531 CAD models in total for the training and validation set. Out of these, 75% of the models were used for training the 3D-CNN and the remaining 25% of the models were used for validation or fine-tuning the hyper-parameters of the 3D-CNN. A detailed description of the training process is provided in Section 2.5. The trained DLDFM network is then tested using the test set CAD models. The test set contains 675 geometries.

2.7.1 Voxelization Timings

We recorded the time taken to generate the volumetric representation of the B-Rep CAD geometries at the voxel resolution used for the training. There is a slight dependence of the voxelization timings on the number of triangles in the CAD model, but it is predominantly dependent on the voxel grid resolution. The voxelization timings for the different CAD models shown in Table 2.4 is shown in Table 2.1. The table also includes the voxelization timings of the two large CAD models that are also used for testing the DLDFM.

2.7.2 Tuning of the Hyper-Parameters

Finding the best hyper-parameters of the DLDFM is very important for the performance of the network. The hyper-parameters of DLDFM are fine-tuned to have least validation loss. Tweaking the hyper-parameters such as the number of layers, number of filters in each layer and filter sizes, the optimizer, loss function etc. is important to obtain the best possible results. Figure 2.7 shows the variation of training and validation accuracy as a function of the number of parameters. The parameters with the best training and validation accuracy is then selected to be the final DLDFM network. Another
Table 2.1: Timings and voxel count of the voxelization of the CAD geometries and large-scale model.

<table>
<thead>
<tr>
<th>CAD Geometry</th>
<th>Triangle Count</th>
<th>Total Voxel Resolution</th>
<th>Inside Voxels</th>
<th>Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a)</td>
<td>61688</td>
<td>262,144</td>
<td>106,240</td>
<td>0.177</td>
</tr>
<tr>
<td>(b)</td>
<td>59036</td>
<td>262,144</td>
<td>108,755</td>
<td>0.154</td>
</tr>
<tr>
<td>(c)</td>
<td>68694</td>
<td>262,144</td>
<td>110,592</td>
<td>0.179</td>
</tr>
<tr>
<td>(d)</td>
<td>63396</td>
<td>262,144</td>
<td>110,238</td>
<td>0.160</td>
</tr>
<tr>
<td>(e)</td>
<td>50544</td>
<td>262,144</td>
<td>72,778</td>
<td>0.150</td>
</tr>
<tr>
<td>(f)</td>
<td>51958</td>
<td>262,144</td>
<td>72,960</td>
<td>0.165</td>
</tr>
<tr>
<td>(g)</td>
<td>46582</td>
<td>262,144</td>
<td>88,293</td>
<td>0.151</td>
</tr>
<tr>
<td>(h)</td>
<td>68704</td>
<td>262,144</td>
<td>110,536</td>
<td>0.153</td>
</tr>
<tr>
<td>Part 1</td>
<td>57914</td>
<td>4,718,592</td>
<td>4,451,177</td>
<td>2.074</td>
</tr>
<tr>
<td>Part 2</td>
<td>96994</td>
<td>9,437,184</td>
<td>6,943,570</td>
<td>6.677</td>
</tr>
</tbody>
</table>

important aspect of the training process is batch normalization, which allows for better training. Without batch normalization, the network saturates and the training reaches a saddle point leading to low training accuracy as shown in Figure 2.8.

A batch size of 64 is selected while training the networks. The training was performed using Keras [4] with a TensorFlow [1] backend in Python environment. The training of DLDFM networks was performed in a workstation with 128GB CPU RAM and a NVIDIA Titan X GPU with 12GB GPU RAM. The training is performed until the validation loss remains constant for at least 10 consecutive epochs. The final architecture of the DLDFM network is described in the Table 2.2. The trained DLDFM network is then tested using the test set CAD geometries, which contains 675 CAD models.

### 2.7.3 Results from Test Data Set

After successful training, the DLDFM network was tested on a test set to benchmark its performance. Accuracy of DLDFM network on the test set using the two data representations is shown in Table 2.3. The test-set has completely different geometries compared
to the training set. Thus, it can be seen that the DLDFM is learning the localized geometric features that directly contribute to the manufacturability or the non-manufacturability of the CAD geometry. Figure 2.10 shows the confusion matrix of DLDFM test results using binary occupancy information and binary occupancy augmented with voxel-wise normal information.

Using the trained DLDFM network, it is possible to obtain the localization of the feature activating the decision of the DLDFM. The 3D-GradCAM renderings for various cases are shown in the Table 2.4. We have used 3D-GradCAM to visualize the results of various inputs such as manufacturable holes, non-manufacturable-holes, multiple holes in same face, and holes in multiple faces of the cube. 3D-GradCAM can localize the features that can cause the part to be non-manufacturable. For example, in Table 2.4, the second example shows a CAD model with a hole, which is non-manufacturable because
Table 2.2: Optimized hyper-parameters with the least validation loss.

<table>
<thead>
<tr>
<th>Common Training Parameters</th>
<th>DLDFM Network</th>
<th>Hyper-parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Batch Size: 64</td>
<td>Voxel</td>
<td>Convolutional Layer 1 (8 filters with size 8×8)</td>
</tr>
<tr>
<td>Optimizer: Adadelta</td>
<td></td>
<td>Max Pooling Layer 1 (sub-sampling of 2×2)</td>
</tr>
<tr>
<td>Loss Function: Cross-Entropy</td>
<td></td>
<td>Convolutional Layer 2 (12 filters with size 4×4)</td>
</tr>
<tr>
<td></td>
<td>Voxel</td>
<td>Max Pooling Layer 2 (sub-sampling of 2×2)</td>
</tr>
<tr>
<td></td>
<td>+Normals</td>
<td>Convolutional Layer 3 (16 filters with size 2×2)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Fully Connected Layer (128 neurons)</td>
</tr>
</tbody>
</table>

it is too close to one of side faces. This is a difficult example to classify based only on the information of the hole. The 3D-GradCAM rendering correctly identifies the non-manufacturable hole and as a result the DLDFM network also predicts the part to be non-manufacturable.

2.7.4 Relationship Between Data Size and Accuracy

We performed an experiment to assess the effect of the number of data samples on training the network. As discussed earlier, the training samples were selected from a set of 675 models, 408 of which were manufacturable. Table 2.3 illustrates the quantitative performance assessment of the DLDFM on test data sets.

Table 2.3: Quantitative performance assessment of the DLDFM on test data sets.

<table>
<thead>
<tr>
<th>Test Data Type</th>
<th>Model Description</th>
<th>True Positive</th>
<th>True Negative</th>
<th>False Positive</th>
<th>False Negative</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>675 models</td>
<td>In-outs</td>
<td>391</td>
<td>90</td>
<td>17</td>
<td>176</td>
<td>0.7136</td>
</tr>
<tr>
<td></td>
<td>In-outs + Normals</td>
<td>334</td>
<td>201</td>
<td>74</td>
<td>65</td>
<td>0.7938</td>
</tr>
</tbody>
</table>
Table 2.4: Illustrative examples of manufacturability prediction and interpretation using the DLDFM framework.

<table>
<thead>
<tr>
<th>CAD Models</th>
<th>3D-GradCam Visualizations</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a) Manufacturable</td>
<td>(e) Manufacturable</td>
</tr>
<tr>
<td>(b) Non-Manufacturable</td>
<td>(f) Non-Manufacturable</td>
</tr>
<tr>
<td>(c) Non-Manufacturable</td>
<td>(g) Non-Manufacturable</td>
</tr>
<tr>
<td>(d) Non-Manufacturable</td>
<td>(h) Manufacturable</td>
</tr>
</tbody>
</table>
more than 800,000 CAD models. We trained the DLDFM network with different number of training samples and recorded the variation in the training and validation accuracy (Figure 2.11). It can be seen that the training accuracy does not vary, while the validation accuracy improves with the addition of more training data. The only limitation in increasing the training data size is computational cost of training from the large data.

In order to compare our DLDFM method with other feature based detection method, we need to assume prior knowledge of the geometry of the feature. We compare the performance of DLDFM with a hole-ratio based feature detection system, since the hole-ratio is the most dominant feature that can be used for feature recognition. This is found in many CAD tools such as SolidWorks [16]. A hole-ratio based manufacturability analysis will fail for other thickness based rules discussed in Section 2.3, which makes it non-reliable. However including such thickness-based checks is not trivial for such a hand-crafted system. The false positive and false negative examples for hole-ratio based system is higher than DLDFM as shown in Table 2.5 thus demonstrating the reliability of DLDFM and its capability to learn multiple complex features, which one would have to hand-craft to use a feature detection algorithm to analyze the manufacturability of a part.
2.7.5 Manufacturability Analysis of a Realistic Part

In order to infer the manufacturability of a large scale realistic CAD model, we use the trained 3D-CNN model (Section 2.5) to infer the manufacturability of small slices of the voxelized geometry. If the initial grid resolution chosen for the training is $l \times m \times n$, then the large scale CAD model is voxelized using the same grid size, resulting in a larger grid resolution, $l_1 \times m_1 \times n_1$. We pad this voxel grid with $l/2$, $m/2$, $n/2$ voxels of zero value in all the directions to correctly analyze the boundaries. We then slice $l \times m \times n$ voxels to obtain one inference; we stride by one in each direction to obtain a map of size $l_1 \times m_1 \times n_1$ inferring the spatial non-manufacturability of the large CAD model. Note that some of the slices having partial coverage of the features will be inferred as non-manufacturable, thus raising a false alarm, but when the hole is bounded correctly by the slice, the inference is comparable to the performance on the previous training data set. As shown in the Figure 2.12 there is a square region surrounding the features identified in the large CAD model, with false non-manufacturability inference. However, the central region in the patch indicates the true manufacturability inference of the feature, i.e. whether the
Figure 2.12: Test results on the large scale CAD model using DLDFM framework to detect manufacturable and non-manufacturable drilled holes. (a) Isometric view of the test case with DLDFM inference mapped to the CAD model for comparison. (b) Top view with the holes aligned with the inference map. Each hole is marked as M - manufacturable or as NM - non-manufacturable based on the reason for non-manufacturability: NM-1 - non-manufacturable due to D/d ratio; NM-2 - Non-manufacturable due to small amount of material in the depth direction; B - borderline cases.
Table 2.5: Comparison of the performance of the DLDFM on the training data with traditional DFM based only on the depth to diameter ratio of the drilled holes.

<table>
<thead>
<tr>
<th>CAD Models</th>
<th>3334</th>
<th>4361</th>
<th>5387</th>
<th>8405</th>
</tr>
</thead>
<tbody>
<tr>
<td>DFM</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hole ratio</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>True Positive</td>
<td>575</td>
<td>684</td>
<td>912</td>
<td>1053</td>
</tr>
<tr>
<td>True Negative</td>
<td>676</td>
<td>868</td>
<td>821</td>
<td>990</td>
</tr>
<tr>
<td>False Positive</td>
<td>165</td>
<td>59</td>
<td>208</td>
<td>67</td>
</tr>
<tr>
<td>False Negative</td>
<td>251</td>
<td>56</td>
<td>239</td>
<td>70</td>
</tr>
<tr>
<td>DLDFM</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hole ratio</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>True Positive</td>
<td>1339</td>
<td>1875</td>
<td>1145</td>
<td>1568</td>
</tr>
<tr>
<td>True Negative</td>
<td>1191</td>
<td>1191</td>
<td>1011</td>
<td>990</td>
</tr>
<tr>
<td>False Positive</td>
<td>73</td>
<td>316</td>
<td>73</td>
<td>316</td>
</tr>
<tr>
<td>False Negative</td>
<td>473</td>
<td>153</td>
<td>90</td>
<td>437</td>
</tr>
<tr>
<td>Training Accuracy</td>
<td>0.7504</td>
<td><strong>0.9310</strong></td>
<td>0.7949</td>
<td><strong>0.9372</strong></td>
</tr>
<tr>
<td></td>
<td>0.8006</td>
<td><strong>0.9394</strong></td>
<td>0.8136</td>
<td><strong>0.9340</strong></td>
</tr>
</tbody>
</table>

Drilled hole is manufacturable or not. According to the color map shown in Figure 2.12, the central blue colored regions inside holes correspond to their manufacturability and likewise the central red colored regions correspond to non-manufacturability. Comparing the inference of the DLDFM with the original CAD model in Figure 2.3 we see that all the green (manufacturable) holes align to the blue regions in the inference and the red (non-manufacturable) holes align to the red regions as expected. The same is true in another example of a complex large part with many holes resembling a injection mold die with cooling holes.

2.8 Conclusion

In this paper, we demonstrate the feasibility of using 3D-CNNs to identify local features of interest using a voxel-based approach. The 3D-CNN was able to learn local geometric features directly from the voxelized model, without any additional shape information. As a result, the 3D-CNN was able to identify the local geometric features irrespective of the external object shape. Hence, a 3D-CNN can be used effectively to identify local features, which in turn can be used to define a metric that may be used for successful object classification. In addition, using the 3D-GradCAM eliminates the black box
notion about CNNs; the DLDFM framework provides feedback about the source of non-manufacturability. The feedback is helpful to understand which particular local feature among various other features in a CAD geometry accounts for the non-manufacturability and possibly modify the design appropriately.

We apply our local feature detection tool to build a deep-learning-based DFM framework (DLDFM), a novel application of deep learning for cyber-enabled manufacturing. To the best of our knowledge, this is the first application of deep learning to learn the different DFM rules associated with design for manufacturing. In this paper, our DLDFM framework was able to successfully learn the complex DFM rules for drilling, which include not only the depth-to-diameter ratio of the holes but also their position and type (through hole vs. blind). The framework can be extended to learn manufacturable features for a variety of manufacturing processes such as milling, turning, etc. We envision training multiple networks for specific manufacturing processes, which can be concurrently used to classify the same design with respect to their manufacturability using different processes. Thus, an interactive decision-support system for DFM can be integrated with current CAD systems, which can provide real-time manufacturability analysis while the component is being designed. This would decrease the design time, leading to significant cost-savings.

2.9 References


CHAPTER 3.  MULTI-LEVEL 3D CNN FOR LEARNING
MULTI-SCALE SPATIAL FEATURES

A paper accepted by IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)

Sambit Ghadai, Xian Yeow Lee, Aditya Balu, Soumik Sarkar and Adarsh Krishnamurthy

3.1 Abstract

Learning multi-scale spatial features from 3D spatial geometric representations of objects such as point clouds, 3D CAD models, surfaces, and RGB-D data can potentially improve object recognition accuracy. Current deep learning approaches learn such features either using structured data representations such as volume occupancy grids (voxels) and octrees or from unstructured representations such as graphs and point clouds. Structured representations are in general restricted by their inherent limitations on resolution such as the voxel grid dimensions or the maximum octree depth. At the same time, it is challenging to learn directly from unstructured representations of 3D data due to non-uniformity among the samples. A hierarchical approach that maintains the structure at a larger scale while still accounting for the details at a smaller scale in specific spatial locations can provide an optimal solution for learning from 3D data. In this paper, we propose a multi-level learning approach to capture large-scale features at a coarse level (for example, using a coarse voxelization), while simultaneously capturing flexible sparse information of the
small-scale features at a fine level (for example, a local fine-level voxel grid) at different spatial locations. To demonstrate the utility of the proposed multi-resolution learning, we use a multi-level voxel representation of CAD models to perform object recognition. The multi-level voxel representation consists of a coarse voxel grid that contains volumetric information of the 3D objects and multiple fine-level voxel grids corresponding to each voxel in the coarse grid that contains a portion of the object boundary. Finally, we demonstrate the performance of our multi-resolution learning algorithm for object recognition. For object recognition, we outperform a number of previously published benchmarks, while using significantly less memory during training.

3.2 Introduction

A three dimensional object comprises of a different multi-scale features inherent to its geometry and its overall shape. Considerable efforts have been made for shape detection & searching [11, 30, 36, 39] and feature detection [7, 21, 38] from 3D objects and CAD models for various applications such as design, manufacturing, and analysis [4, 12, 13, 19, 27]. A major research area in computer vision is dedicated to representing 3D spatial data and extracting meaningful information or features from it using deep neural networks. Previous works have proposed learning from 3D data by projecting the 3D information to 2D or 2.5D (depth inclusion) images, thereby extending image recognition principles to 3D object recognition [34, 35, 43]. In addition, researchers have demonstrated object recognition from multiple 2D views of the 3D object [14, 20, 25, 32]. Though this approach is effective in many applications including 3D reconstruction, some spatial relationships among the features may be lost and this makes it infeasible for certain problems such as graphics rendering [37], point cloud labeling [26], design and manufacturing [9] etc. However, a major limitation of directly using 3D data with deep neural networks is the
high memory requirement. The presence of abundant information in spatial data coupled with large data requirement for efficient training of deep learning algorithms render this task impractical for high-resolution 3D data. On the other hand, since the data representation resolution directly influences the memory requirement, reducing memory usage results in low fidelity of feature extraction and detection. Hence, new efficient and scalable feature learning techniques are required to deal with detection, classification, and feature analysis of CAD models.

Effective utilization of high resolution 3D CAD data for object classification using deep learning requires development of new data representation as well as novel deep learning architectures. The learning algorithm needs to preserve spatial localization while learning hierarchical features from data. Therefore, convolutional neural networks (CNNs), which have been proven to be effective for 3D spatial data [10, 24] are the natural candidates. However, training CNNs using uniform data representations (such as voxelized representation) become inefficient when the spatial features exist on different scales since the uniform data representation cannot effectively accommodate this non-uniformity [1]. This is illustrated in Figure 3.1, where the CAD models have features such as holes, slots, and pockets pertaining to different spatial scales. Thus, there is a need to represent the CAD data in a hierarchical manner along with building an efficient deep learning architecture to learn features from each level of the hierarchy. In addition, the data representation needs to be both memory efficient and preserve the spatial relationship in the data. Similar to the learning paradigm of CNNs, where the first few layers learn fine-level features and final layers learn coarse-level features, hierarchical learning can enable extraction of key features from a fine level of the hierarchy and then merge the understanding of the small scale aspects with the coarse level features. Such an approach will also exploit the sparse nature of the spatial data better. Extending this idea explicitly, we make use of a
Figure 3.1: An illustrative example showing the need for multi scale feature detection in two mechanical parts (CAD models). The CAD models have a multitude of features, each on a separate spatial scale, which create challenges in specific feature detection.

multi-level voxel representation of CAD models that can better represent 3D models in a memory efficient manner.

One of the primary requirements for the efficient functioning of a deep neural network, especially CNNs, is a large representative labelled dataset that facilitates object classification. Such a large representative dataset of 3D CAD models, that has embedded physical features for engineering applications, is unavailable to the best of our knowledge. So to demonstrate the main concept of our paper, we use the open source ModelNet dataset [43], which is a large collection of 3D models, to perform a object detection and classification task based on the features of each class of models in the dataset. This follows along the concept that 3D objects can be segregated based on their feature-based similarity [18], as well as geometry-based similarity [15, 31].
In this paper, we present a novel approach to enable hierarchical learning of features and thus perform object detection, from a flexible multi-level voxel representation of 3D CAD data using deep learning. We achieve this by adopting the multi-level voxelization framework developed by Young and Krishnamurthy [45], that generates a multi-level voxel grid structure. The 3D object is represented using a binary occupancy grid at two levels with independent user-defined resolutions at each level. Extending the same idea to the deep learning algorithm, we have developed a multi-level CNN that can efficiently learn from this multi-level data representation. In addition, we have devised an interpretability based feedback approach to enable the flow of salient information from the fine level to the coarse level.

The specific contributions of this paper include:

1. A framework for learning from hierarchical multi-scale representation of 3D CAD data, where there are two levels of information, a coarse level and a fine level. The coarse level description of features in the spatial data can give an overall understanding of the object, while some important finer level features could improve the learning by augmenting the coarse level description.

2. A new method of connecting the multiple learning levels via salient information flow between them. We develop the connection between the two levels of the network using feedback from the coarse level to the fine level provided by interpretability mechanisms such as GradCAM (Gradient of the class activation map) regarding important spatial regions.

3. Demonstration of the above mentioned methods on a multi-level voxel representation used for the purpose of object recognition of ModelNet10 and ModelNet40 datasets. We achieve similar accuracy of results as previously reported, while using considerably lower memory during training.
Table 3.1: Comparisons between different spatial deep learning approaches. Our approach (MRCNN) retains some of the advantages OctNet of such as memory efficiency while still having a flexible data structure. MRCNN also enables spatial multi-scale learning from multi-resolution data.

<table>
<thead>
<tr>
<th>Method</th>
<th>Data Representation</th>
<th>Data Structure</th>
<th>Memory Efficient</th>
<th>Spatial Multi-scale Learning</th>
</tr>
</thead>
<tbody>
<tr>
<td>VoxNet</td>
<td>Voxel</td>
<td>Structured</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>PointNet++</td>
<td>Point cloud</td>
<td>Unstructured</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>OctNet</td>
<td>Octree</td>
<td>Structured</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>MRCNN</td>
<td>Multi-level voxels</td>
<td>Structured-flexible</td>
<td>yes</td>
<td>yes</td>
</tr>
</tbody>
</table>

The paper is arranged as follows. In Section 3.3, we discuss a few relevant works in the field of learning from 3D data using different data representations. In Section 3.4, we describe the multi-resolution representation of 3D data using multi-level voxelization, which we generate using a GPU accelerated voxelization algorithm. We explain the Multi-Resolution Convolutional Neural Network (MRCNN) architecture, that effectively learns the features from the multi-level voxel representation, in Section 3.5. Finally, in Section 3.6, we present the results from evaluating the MRCNN on multi-level voxel data to classify objects in the ModelNet10 dataset and also explain the effectiveness of MRCNN to learn from 3D data with reduced memory usage.

### 3.3 Related Work

Learning from spatial data has been an active research topic and several approaches have been developed by researchers to address this challenge. Most of the approaches can be categorized into two main learning methodologies. The first category of approaches are based on learning from unstructured spatial data such as point clouds [5, 16, 26, 29], meshes [2, 8, 23] and graphs [3, 17, 40]. The second class of methods use structured spatial data such as voxel grids [22, 24, 42–44], octrees [28, 37, 41], RGB-D images [6, 33], etc. Using general deep learning methods (such as CNNs) to learn from unstructured spatial
data is challenging, since many of the operations require a structured input. Our work is focused on a sub-class of structured methods that use volumetric representation (voxels, octrees) to learn from spatial data, while also being flexible in terms of the data structure. Our multi-level voxel data structure makes use of user-defined voxel resolution at each level, making it more flexible than the octree data structure (each voxel is divided exactly into $2^3$ sub-voxels). This allows us to achieve very high effective resolutions using only two levels, while retaining the memory efficiency.

### 3.3.1 Multi-level Voxel Learning

Learning from 3D voxel data was initially explored by Wu et al. [43] (3DShapeNet) using convolutional deep belief networks (CDBNs) and later by Maturana and Scherer [24] (VoxNet) using 3D-CNN. It is challenging to achieve a good classification accuracy on the ModelNet10 and ModelNet40 datasets using these approaches. Most approaches that use voxel data use a maximum of $32^3$ resolution; recently with the increase in GPU memory, using a resolution of $64^3$ is possible. However, increasing the resolution beyond that is not practical with current systems for training deep networks using dense voxel grids. Increasing the effective voxel resolution beyond $64^3$ require new specialized data structures. Tatarchenko et al. [37] and Riegler et al. [28] developed OctNets, which make use of an octree-based voxel representation of the data.

In a tree structure each voxel is represented as a node and each of those nodes are connected to exactly eight subdivided voxels or octants. However, traversing the octree structure requires recursive algorithms. OctNets solved this problem by making use of a shallow octree data stored using a regular grid. This allowed them to directly index the data in the octree without recursively parsing the tree and were able to achieve an effective resolution of $256^3$. 
Figure 3.3: Multi-level voxelization of B-rep CAD models. The fine level voxelization is performed only near the boundaries of the coarse level voxelization. The final resolution is equivalent to having dense level voxels throughout the model.

In this work, we make use of a different strategy of having multiple levels of voxelization that can have arbitrary resolutions at each level. Using this approach, we can also achieve very high resolution similar to those achieved using OctNets. For example, we can achieve effective resolution of $256^3$ by having a coarse resolution of $64^3$ and a fine resolution of $4^3$.

### 3.4 Multi-level Voxelization

In this section, we briefly describe the GPU-accelerated algorithm [45] we used to generate a multi-level voxelization of a boundary representation (B-rep) CAD model. The multi-level voxelization is a binary occupancy grid having two major components namely, *coarse-level voxelization* and *fine-level voxelization*. Multi-resolution voxel representations of B-rep CAD models are shown in Figure 3.3 along with corresponding coarse and dense voxel grids. At each level, the voxel occupancy is represented using a binary value of 0/1 that defines whether a voxel is occupied by the object or not. To create the coarse-level voxelization, a standard grid of voxels using an user-defined resolution is
constructed in the region occupied by the object (denoted by the axis-aligned bounding box (AABB)). A triangle-box intersection test is used in the next step to identify the boundary voxels by checking the intersection of each triangle of the B-rep model with every voxel. Identifying the boundary voxels enable further division of the coarse-level grid into a fine resolution only at the boundary of the object without cubically increasing the voxel count. Once the boundary voxels are identified, they are further sub-divided to construct the fine-level voxel grid only at the boundary locations. The same triangle-box intersection test is then used to identify the fine-level voxels that intersect with the triangles of the B-rep model. Further, an index array is created using exclusive prefix sum that maps the memory location of each coarse-level boundary voxel to its corresponding fine-level voxel grid as shown in Figure 3.4. The complete voxelization framework is GPU-accelerated, where each triangle-box intersection test is computed in parallel at individual levels. Using this method on the GPU, a high-resolution (effective resolution
of 128³) multi-level voxelization of a low-mid polygon count model can be generated in about 180ms.

Representing a 3D model using a multi-level voxel grid structure exploits the sparse nature of the data. When spatial data is represented using voxel grids, the voxel count increases as $O(n^3)$ with increase in resolution of the grid and in turn reduces the total voxel space occupied by the object. To prove this, we represent the coarse-level voxel grid by $G_c$ and its corresponding fine-level voxel grid by $G_f$ with resolutions of $n_c \times n_c \times n_c$ and $n_f \times n_f \times n_f$, respectively. Let the number of boundary voxels be $\phi_b$. Comparing it with a dense voxel grid of resolution $n_d \times n_d \times n_d$ where $n_d = n_c \times n_f$, we see that, total voxels in multi-level data structure is $\phi_b \times n_f^3$. As the number of boundary voxels can never be more than the total voxel count, $1 \leq \phi_b \leq n_c^3$. Define

$$G_c \rightarrow n_c \times n_c \times n_c$$
$$G_f \rightarrow n_f \times n_f \times n_f$$
$$n_d = n_c \times n_f$$

From the above,

$$1 \leq \#\text{Boundary Voxels}, \phi_b \leq \text{Total Voxels}, n_c$$

$$\therefore \phi_b \times n_f^3 \leq n_c^3 \times n_f^3 = n_d^3$$

$\Rightarrow$ Voxel count\text{multi} $\leq$ Voxel count\text{dense}.

The number of boundary voxels for a fixed-genus closed object scales with $O(n^2)$ with the resolution $n$. This scaling, in practical cases, is much smaller than the volumetric scaling of voxels, leading to a compression in the data structure without losing any accuracy in the representation. In addition, the spatial locations of the locally dense fine level voxels can effectively aid in learning the shape of the object in the coarse level.
Figure 3.5: Multi-Resolution Convolutional Neural Network (MRCNN). Our proposed network can learn from a hierarchical CAD data representation with a coarse level of information and information of boundary voxels which connects to the fine level voxels. For a forward pass (left to right) the information learnt from selected fine level voxels using the L2 CNN is embedded in the coarse level input to L1 CNN and then the final prediction is obtained. The backward pass follows the reverse order of the forward pass (right to left).

3.5 Multi-resolution CNN

In the previous section, we showed that representing 3D spatial data using a flexible multi-level data structure exploits the sparse nature of the data and is more memory efficient. The multi-level voxel data structure stores information pertaining to the geometry of an object in two hierarchical levels and hence, an optimal learning algorithm should use features from both levels. In this section, we explain the proposed multi-resolution hierarchical learning algorithm that enables end-to-end learning of individual features from both the coarse and fine voxel grids. Further, we investigate an interpretability based feedback mechanism for adaptively extracting voxel zones that bridge the learning from the fine to the coarse level.
The multi-resolution convolutional neural network (MRCNN) consists of two CNNs, with kernels performing 3D convolution operations, for learning the features in each of the voxel levels. One of these 3DCNNs, named as Coarse-level CNN, takes in the coarse level voxels as input, and the other CNN called Fine-level CNN accepts the fine level voxels as input. Let $\theta_1$ be the set of weights for the Coarse-level CNN and let $f(x, \theta_1)$ be the predicted output for a given coarse-level input $x$ and $\theta_1$. This provides a benchmark of the network's performance with some loss $\epsilon$ in the prediction. We then augment the performance of the network by adding Fine-level CNN with another set of weights, $\theta_2$, which are used to learn features from the fine level resolution. These two neural networks are intelligently combined to work together as a single unit in both forward pass and backward pass of the algorithm. This facilitates optimal learning from the multi-level data representation and makes it computationally comparable to performing sparse convolutions. The forward pass and backward pass of the proposed network are explained in the next two sections. The complete operation of MRCNN is explained schematically in Figure 3.5.

3.5.1 Forward Computation of MRCNN

Recalling the voxel data representation, a 3D object is represented as a grid of coarse voxels (say $8^3$ resolution) with a binary voxel value of 0/1 at the boundary of the object and each of those boundary voxel (with coarse voxel value of 1) are further subdivided into a fine voxel grid (say $4^3$ resolution) with similar binary values 0/1.

The forward computation of MRCNN starts from the fine-level voxel grid by random sampling a subset, $\phi$, of the total boundary voxels, $\Phi$, of a 3D model. Each of these $\phi$ boundary voxels, with individual fine voxel grid $\vartheta_2$ of resolution $n_f^3$, are used as input to Fine-level CNN. The Fine-level CNN network consists of blocks of convolution - max pooling layer pairs and fully connected layers connected in conjunction, each with a ReLU function.
Algorithm 1: MRCNN Forward & Backward Passes.

Forward Pass:

1. forall Boundary voxels, $\phi_b$ in parallel do
2. \hspace{1em} $\eta_b = \text{Forward}_{\text{Fine-levelCNN}}(\vartheta_2_b)$;
3. \hspace{1em} $\vartheta_1(\phi_b) = \eta_b$
4. end
5. $y_{\text{pred}} = \text{Forward}_{\text{Coarse-levelCNN}}(\vartheta_1)$

Backward Pass:

6. $d\vartheta_1 = \text{Backward}_{\text{Coarse-levelCNN}}(\vartheta_1, dy_{\text{pred}})$
7. forall Boundary voxels, $\phi_b$ in parallel do
8. \hspace{1em} $d\eta_b = d\vartheta_1(\phi_b)$
9. \hspace{1em} $d\vartheta_2_b = \text{Backward}_{\text{Coarse-levelCNN}}(\vartheta_2_b, d\eta_b)$
10. end

associated with it. *Fine-Level CNN* outputs a single real numbered value $\eta_b \forall b \in \phi, \eta_b \in \mathbb{R}$. These set of $\eta_{\phi_b}$ values are sent forward to be combined with coarse-level voxel grid, $\vartheta_1$. The outputs of *Fine-level grid*, $\eta_b$, replace the original coarse voxel grid values at the appropriate voxel positions, $\vartheta_1(\phi_b)$. This is performed with the help of the index array that maps the position of each coarse-level boundary voxel with its respective fine-level voxel grid.

In the corresponding phase of forward computation of MRCNN, the coarse-level voxel grid with selective embeddings from *Fine-level CNN*, $\vartheta_1$, is used as an input to the *Coarse-level CNN*. The architecture of *Coarse-level CNN* network comprises of different permutations of convolution - max pooling layer blocks. The end section of the network has multiple fully connected layers and the output is the class prediction probability vector. Categorical cross-entropy loss function is used to compute the loss of the predicted classes with the true class labels. The forward run of MRCNN network algorithm is depicted in Algorithm 1. Network hyperparameter details are provided in the supplementary section.
3.5.2 Backward Computation of MRCNN

While the forward computation of MRCNN may be trivial, the back-propagation of the MRCNN can be tricky to correctly implement. The main challenge of the back-propagation computation is to link the two networks such that the gradients can passed on from the coarser level network to the finer level network. Without this link, the weights of the finer level network wouldn’t be updated accordingly. The final loss between the \( y_{\text{pred}} \) and \( y_{\text{true}} \) of the coarse level network is first computed using categorical cross-entropy loss. Back-propagating this loss through the coarse level network is performed using the traditional back-propagation method where the gradient of \( y_{\text{pred}} \) from the loss \( L \) is back-propagated all the way through the intermediate layers of the network and finally to the input data to obtain their respective gradients. Let the gradient of the loss with respect to coarse input be \( d\theta_1 \), using the prefix sum, we track the gradients of the outputs of fine level network and use it to back-propagate through the network to obtain the gradients of fine level network. This process is explained in Algorithm 1 and Equation 3.1 shows the back-propagation of the loss from the output level to the fine level network.

\[
\frac{\partial L}{\partial \theta_2(\phi)} = \frac{\partial L}{\partial \theta_1(\phi)} \frac{\partial \theta_1(\phi)}{\partial \theta_2(\phi)} \tag{3.1}
\]

It is also worthwhile to note that since the same Fine-level CNN is shared among all the boundary voxels, the gradients of \( \theta_2 \) for Fine-level CNN are computed for all boundary voxels only once. With that gradient, the network could be trained to update its weights \( \theta_1 \) and \( \theta_2 \) in such a way that the loss \( L \), of the final prediction \( y_{\text{pred}} \) is minimized. This facilitates the end-to-end learning of network parameters \( \theta_1, \theta_2 \) in a hierarchical order. The network parameters’ update could be performed using any optimizer, such as SGD, Adam, Adadelta, etc.
3.6 Experimental Results & Discussion

In this section, we present the classification results of the proposed MRCNN framework on Princeton’s ModelNet10 dataset [43] that contains 3D CAD models of 10 different categories. The 3D CAD models were voxelized using the voxelization scheme mentioned in Section 3.4, yielding a set of coarse voxel grid of $8^3$ and another set of dense voxel grid of $32^3$. In addition, we also voxelized 2 sets of multi-resolution data to test the efficacy of MRCNN; a $8^3$ coarse voxel grid with a $4^3$ fine voxel grid giving an effective resolution of $32^3$ resolution and a $32^3$ coarse voxel grid with a $4^3$ fine voxel grid, resulting in an effective resolution of $128^3$. We ran a set of experiments on the 4 different resolutions of the data and compared the classification performance between a Coarse-Level CNN applied on the coarse and dense resolution data and MRCNN applied on the multi-resolution data. For the coarse and dense resolutions, we apply the Coarse-level CNN on the data and evaluated the classification performance of the network. For the multi-resolution representation, we applied our proposed MRCNN by sampling 40% of SVZ, computed by 3DGradCAM, and allowed the fine resolution voxels of these coarse voxels to run on Fine-level CNN. We selectively embed these values in the coarse level boundary voxels and continue the forward pass as explained in Section 3.5.1. We find that sampling 40% of boundary voxels gives a good classification performance using MRCNN without prolonging the training time excessively. To demonstrate that our proposed framework is agnostic to the network’s architecture, we ran multiple sets of experiments across all four resolutions with varying network architectures that are extensively described in the supplementary section.

Figure 3.6 shows the mean test accuracy obtained from the experiments with the variance of the accuracy represented by the shaded regions. There is a clear trend which shows that a denser resolution results in a better classification accuracy. Starting from the
Figure 3.6: Mean classification performance with different input resolutions on ModelNet10 dataset. The respective input voxel resolutions are mentioned along the x-axis. The coarse and dense resolutions are trained with a conventional 3DCNN while the multi-resolution voxel grids are trained with MRCNN.

coarse resolution of $8^3$, the *Coarse-level CNN* is able to achieve a mean test accuracy of 81%. However, with a multi-resolution voxel grid of $32^3$ effective resolution, MRCNN is able to obtain a better mean classification accuracy. Subsequently, a regular CNN applied on a dense voxel grid of $32^3$ is able to achieve a slightly better classification accuracy than both. Due to memory constraints of GPUs, we are unable to demonstrate the performance of a *Coarse-level CNN* applied on a dense resolution data of $128^3$. Nonetheless, using MRCNN, we are able to achieve the best classification performance using an effective resolution of $128^3$ represented by a multi-resolution voxel grid with a base resolution of $32^3$ and a finer resolution of $4^3$.

A comparison of our results with the performance of other spatial deep learning methods is also tabulated in Table 3.2. We compare our performance with OctNet due to the similarities in data representation (high resolution voxel grid) and classification task that
Table 3.2: Comparison of deep learning frameworks with voxel based representation for ModelNet10 object recognition. MRCNN (our method) along with OctNet can learn from a voxel grid resolution of $128^3$, so we compare our work with OctNet for better clarification. Also we show the performance of other such voxel based spatial learning methods. * represents value interpreted from plot.

<table>
<thead>
<tr>
<th>Method</th>
<th>Data Representation</th>
<th>Accuracy %</th>
</tr>
</thead>
<tbody>
<tr>
<td>MRCNN</td>
<td>Multi-level voxels</td>
<td>91.3</td>
</tr>
<tr>
<td>OctNet [28]</td>
<td>Octree Voxels</td>
<td>91.0*</td>
</tr>
<tr>
<td>3D Shapenets [43]</td>
<td>Voxels</td>
<td>83.5</td>
</tr>
<tr>
<td>VoxNet [24]</td>
<td>Voxels</td>
<td>92.0</td>
</tr>
<tr>
<td>Beam Search [44]</td>
<td>Voxels</td>
<td>88.0</td>
</tr>
<tr>
<td>3DGAN [42]</td>
<td>Voxels</td>
<td>91.0</td>
</tr>
<tr>
<td>binVoxNetPlus [22]</td>
<td>Voxels</td>
<td>92.3</td>
</tr>
<tr>
<td>LightNet [46]</td>
<td>Voxels</td>
<td>93.9</td>
</tr>
</tbody>
</table>

exploits the sparsity in spatial data. In addition to that, we compare MRCNN performance with other voxel based methods employed on the ModelNet10 dataset. We can see that MRCNN (91.3%) outperforms some of the voxel based methods and is better at classification than OctNet (91.0%). To compare with VoxNet, our dense-level network with selectively chosen hyperparameters has a better performance (92.1%) than VoxNet (92.0%) as can be seen from Figure 3.6, while consuming memory comparable to coarser voxel grid.

An interesting observation is that, although the mean classification accuracy achieved on coarse voxel grid is relatively high, there is a great variance across the set of experiments performed as well, with certain network architectures having a accuracy of only 55%. Generally, we observed that network architectures that uses average-pooling layers sees a bigger increase in performance when used with MRCNN. Hence, the advantages of MRCNN is two-fold. First, we show that with a multi-level data structure, MRCNN is able to learn simultaneously from the two levels of data and performs better than a standard CNN trained on coarse resolution data. Secondly, MRCNN greatly reduces the
Figure 3.7: GPU memory usage of MRCNN training & equivalent CNN training on specified voxel grid resolutions. Red horizontal line shows the current prominent GPU capacity. Blue hatched bar depicts the anticipated memory usage while training a $128^3$ dense voxel grid on CNN.

Effect of varying network architectures on the performance of the network, especially on the lower bound of the performance. This is advantageous as it allows greater flexibility when designing a deep network.

An additional advantage of the MRCNN framework is the GPU memory requirement during training of the network. In Figure 3.7, we show a comparison between the memory requirements of the GPU for training on the four different resolutions of data. As shown in Figure 3.7, the memory requirements of the GPU scales exponentially with the test accuracy due to which we were unable to train a dense-level network on $128^3$ voxel resolution (shown as a blue hatched bar). To achieve a 6% increase in test accuracy by training on dense resolution data, an additional GPU memory of at least 300% is required compared to training using MRCNN. This highlights the effect of sparsity by dense resolution voxel grid where the increase in classification performance scales non-linearly with data resolution.
3.7 Conclusion

In this paper, we explore a novel deep learning architecture, MRCNN to learn from 3D data in an hierarchical manner using multi-resolution voxel-based data structures. We also demonstrated that a model interpretability mechanism can be leveraged to enable information flow between the two levels of MRCNN. Our object recognition results show that MRCNN performance is significantly better and robust compared to that of the regular CNNs trained on coarse-resolution data while having similar memory requirements. MRCNN also performs almost as well as CNNs trained on dense data with equivalent resolution while keeping the memory requirements significantly lower. This inference can be extended to feature rich CAD models of mechanical components given the availability of such a large dataset of models. Future works will include exploring efficacies of MRCNN on such a collection of CAD models and various object recognition datasets as well as other relevant engineering problems such as design, manufacturing and analysis, where extraction of multi-scale features is critically important.

3.8 References


CHAPTER 4. GPU-ACCELERATED COLLISION FREE NAVIGATION OF VEHICLES IN ENCLOSED SPACES

A paper to be submitted in *IEEE Transactions on Visualization and Computer Graphics*
Sambit Ghadai, Harshil Shah, Alex Schuster, Ivan Thomas, Nathan Greiner and Adarsh Krishnamurthy

4.1 Abstract

We present a framework for navigation of vehicles in large enclosed spaces represented as point clouds using GPU-accelerated collision detection between voxels and point clouds. Our method takes a CAD model of a vehicle, converts it to a volumetric representation or voxels, and computes the collision of the voxels with a point cloud representing the environment to identify a collision-free path for navigation. We perform an adaptive and efficient collision of voxels with point cloud without the need for mesh generation. In addition, we perform a clearance analysis of the navigation and provide theoretical guarantees using a GPU-accelerated voxel-based Minkowski sum algorithm.

4.2 Introduction

Vehicle motion in confined environments plays a major role in the design of both the vehicles and environments. While designing large enclosed spaces, free and efficient movement of vehicles operating in the environment enhances the tasks’ productiv-
ity. Enclosed environments such as warehouses, factory floors, tunnels, and transportation carriers such as ships require design precision and motion planning to achieve high throughput and avoid collisions with the vehicles operating in them. In general, vehicle collisions with their environment are performed in real-time with sensors data to aid in navigation. However, it is not feasible for design purposes to check collision in real-time using physical sensors as redesigning and re-modeling the environment becomes costly and inefficient. Hence, prior knowledge of the vehicle’s interaction with its environment is required to make reliable design decisions. In this paper, we develop an analysis and visualization framework to perform fast collision detection of vehicles with point cloud representations of their environment that provide highly accurate feedback to make efficient design choices. We use a voxelized representation of vehicle CAD models to perform GPU-accelerated collision computation of voxels and point cloud data. We also develop a voxel-based Minkowski addition algorithm that provides flexibility to address design changes on-the-fly based on user-defined collision constraints.

Voxels are volumetric elements that represent a CAD model using occupancy information in a 3-dimensional grid. This results in the discretization of the space occupied by an object into a structured, regular grid. In contrast, general boundary representation (B-rep) schemes for CAD models such as triangular meshes and parametric surfaces represent a model using 2-dimension topological elements (triangles or surface patches). For collision detection using B-rep models, intersection tests are performed between the surface elements to determine the collision. However, this operation is computationally expensive, and checking collision between every element of two or more objects becomes tedious. Further, computing the collision between a point and surface element does not provide accurate information regarding the collision of a point cloud with a CAD model due to gaps and self-intersections between the model’s surface elements.
Voxels, on the other hand, provide a well defined discretized structure to approximate the collisions. Due to voxels being volumetric elements rather than surface elements, computing the point-membership classification of a point with a voxel can be computed easily and is a highly parallel operation. In addition, depending upon the voxel grid resolution or the number of voxels used to represent the model, approximation of collision computation with another object can be easily controlled. In this paper, we compute the collisions between a large point cloud (∼ 1 billion points) with a voxel representation of a CAD model using GPU-acceleration. We perform a boundary voxelization of the B-rep CAD model of a vehicle using a voxelization scheme developed by [29] to classify the object’s boundary with binary occupancy information. Then we perform GPU-accelerated collision detection between the environment, represented as point clouds, and the vehicle voxel model to determine the navigation feasibility of the vehicle in the enclosed environment.

In a dynamic confined environment with vehicle navigation, a tight collision of the vehicle with its immediate environment is not a realizable criterion. In most scenarios, a vehicle needs a sizeable optimal clearance to move within the environment freely. This is essential for a designer while designing the environment and the vehicle or its part segments. However, an exact collision computation between two objects does not provide the necessary information regarding a certain clearance requirement. To facilitate this, we develop a Minkowski addition operation using voxels to create an envelope around the vehicle to be used as the required clearance. Minkowski addition is the set addition operation applied to each element of two sets of points representing shapes in the Euclidean space. We perform the Minkowski addition of a vehicle’s voxel model with another rectilinear voxel grid that conforms to the required clearance value in each orthogonal direction. Then we perform the collision computation between the environment point cloud and the voxel model resulting from Minkowski operation. This provides a flexible frame-
work for the designer to verify the vehicle’s operability with different clearance values. To complement this, we also provide theoretical guarantees on the degree of collision possibility of the vehicle with its environment.

In this paper, we have developed a framework to perform collision detection of vehicles in large enclosed environments. We have developed a GPU-accelerated algorithm to perform collision detection of point clouds and voxel models with additional clearance from our Minkowski sum operation on voxel grids. Our main contributions include:

- A GPU-accelerated static collision detection method of CAD voxel representation with large point clouds to compute and detect collision regions.
- A GPU-accelerated Minkowski addition operation on voxel models to generate a required clearance region around a vehicle model.
- Theoretical guarantees and clearance analysis of vehicles in the point cloud environment.

We first voxelize the CAD model of a vehicle to represent boundary voxel occupancy information in a regular voxel grid in Section 4.4. Then in Section 4.5 we perform the GPU-accelerated collision detection between the vehicle voxel model and a point cloud of the environment to determine the regions in the environment and vehicle responsible for the collision. In Section 4.6, we discuss the theoretical guarantees of the collision and provide an analysis of the clearance between the colliding object with the help of our voxel-based Minkowski addition operation. Finally, we show some examples of the collision detection of different vehicles operating in point cloud based enclosed environments in Section 4.7.
4.3 Related Work

Collision detection or interference detection of objects is an essential segment of many fundamental research areas such as design, manufacturing, computational geometry, robotics, motion planning, and automation [13, 18]. It enables the detection of geometric contacts made between different objects in simulation, which helps avoid, redesign, or re-plan of the operations. Collision detection is generally characterized based on the application area or the type of geometric entities used. With the recent trend in representing solid objects with point clouds, many works have performed geometric queries of point cloud sets. A segment of these works performs collision detection by intersecting rays with surface element representation of points [1, 2, 24]. Distance metric based collision detection has also been used in point clouds to find out objects at a minimum distance [7, 20].

A plethora of works has been contributed to boundary volume hierarchy (BVH) based methods for collision detection using octrees, kd-trees, and other tree implementations [3, 14, 21, 23, 28]. However, BVH methods for broad-phase collision computations provide a general yes/no result for the collision. These methods generally deal with real-time point cloud data streams that guide in tree creation of point sets. For our application, constructing a tree on a vast point cloud (∼1 billion points) creates unnecessary computation overhead. In this paper, we use the concept of BVH to identify a subset of points in a region, depending on the spatial location of the object or vehicle. Then we perform collision detection of the point subset with respect to the voxel representation of the vehicle.

Voxels have recently been more common for collision detection for robotic manipulation, motion planning, and computer graphics. Addressing the \( \binom{n}{2} \) problem, voxels have played a major role in reducing the time complexity of computing the intersection between \( \binom{n}{2} \) pairs of objects [11, 12, 15]. Voxel-based collision detection algo-
Algorithms are especially highly parallelizable due to the discrete representation property of voxels where each thread of computation handles single voxel computations [29]. In the interaction between voxels and other data representations, one type of data structure is converted to another to facilitate homogenous data representation [12, 30]. In contrast, a few methods compute the distance between voxels and points or triangles to facilitate collision detection [11, 19]. However, computing distances and intersections between voxels and other elements present a bottleneck in these methods. In this paper, we facilitate the interaction between a point and a voxel grid by performing a point membership classification of a point with a unit voxel based on its index in the grid. This allows us to find the voxel regions contributing to the collision. Further, we perform a Minkowski addition operation on voxels that preserves the model topology and creates an extra layer of voxels around the original model’s boundary. With the help of voxels, we can perform fast operations on both the methods using GPU-acceleration.

Minkowski sum or Minkowski addition is the element-wise addition of two sets of points in the Euclidean space. Minkowski sum is an essential operation for many engineering applications such as motion and path planning, collision detection, solid and geometric modeling, and computer graphics [5, 17, 22, 26]. Computing the Minkowski sums of convex polyhedra [4, 8, 9] are generally easier than non-convex polyhedra due to very high number of polygon combinations involved [25, 27]. For non-convex polygons, each convex segment of the polygon is first extracted, and a convex polygon-based Minkowski sum operation is performed on each extracted pair to get the final result. Another method used for non-convex polyhedra is to perform a convolution operation on the surface elements and then filtered them to obtain the Minkowski sum boundary. To overcome these limitations of these compute-heavy operations, a voxel-based method [16] is used to generate a voxelized Minkowski sum using rendering based depth textures on the GPU. However, this method excludes any enclosed voids in the polyhedra due to
the flood-fill algorithm used to generate the voxelized representation. In this paper, we
overcome this issue by directly using voxelized polyhedra representations to generate a
voxelized Minkowski sum using 3D convolution techniques. This conforms to our use
of voxels for collision detection as well as Minkowski sum computation. In addition, we
perform fast 3D convolution on voxel grids using GPU-acceleration. Our primary goal of
computing Minkowski sums is to generate a voxel offset layer around the vehicle model
boundary that provides a clearance while detecting collisions.

4.4 Voxelization

In this section, we describe the voxelization process presented in [29] that we have
used to voxelize a 3D CAD model (B-rep) into a regular voxel grid. First, we tessellate
a B-rep model of the vehicle into a triangle soup or a triangle mesh. Then we voxelize
the triangle soup to generate a voxel grid having the information regarding the boundary
and the inside of the model.

To voxelize a triangle soup, a grid of voxels is first constructed from the axis-aligned
bounding box (AABB), bounding all the triangles of the triangle mesh. Using a rendering-
based approach, we perform a point-membership classification of the voxel centers to
classify the voxels as either inside or outside the solid model. A slice-by-slice clipping
is performed on the rendered CAD model. The voxels are classified as inside or outside
based on the odd-even test of the clipped model’s pixels. This process is repeated for all
the voxel slices in the grid, and the final inside-outside information is generated. Scalar
values of 0 and 1 are assigned to the voxels to classify them as inside or outside of the
CAD model, respectively.

To generate the voxelization for the boundary of the solid model, each voxel in the grid
is classified based on whether it contains any triangle or part of a triangle of the object
triangle soup. This is done in two steps. First, the voxels containing the triangles’ vertices are classified as boundary voxels and culled out from the rest of the empty voxel grid. In the next step, the intersection is checked between the triangles and the bounds of each remaining voxel, and the intersecting voxels are marked as boundary voxels using a scalar value of 0.5. The triangle-box intersection test is performed using the separating-axis test [10] and is parallelized using GPU. Details of the algorithm and GPU implementation is explained in [29].

Figure 4.1 shows the voxelized representation of a CAD model of a humvee rendered with an overlapped tessellated representation of the model. This voxel representation of the vehicle is the basis of the collision computations with the point cloud of the environment, as explained in Section 4.5.
4.5 Collision Detection

We describe the point cloud-voxel collision method in this section. After the model’s voxelization, we place the voxel grid in the point cloud at the location where the collision is to be computed. We have the information of the axis-aligned bounding box (AABB) of the voxel grid, which we use as a bounding volume to select a subset of the points in the point cloud. Then we perform a point membership classification of the point with each of the voxels representing the vehicle for defining the point-voxel collisions. We then repeat this process for all the keyframes in a pre-selected path to check for the vehicle’s maneuverability in the enclosed point cloud space.

4.5.1 Axis Aligned Culling Region

Since the point cloud of the environment consists of many points and the vehicle occupies a relatively small region inside the point cloud, we first pre-process the environment into chunks of point cloud information in a grid-like fashion. For further collision computation, a chunk, consisting of the points $P_c$, is selected based on the vehicle’s location in the point cloud environment. Then the extents of the axis-aligned bounding box (AABB) of the vehicle, $V_{AABB}$, are compared with each of the points in $P_c$ to check for occupancy and isolate the points $P_{AABB}$ that is inside $V_{AABB}$. This culling process for the point cloud reduces the number of point-voxel checks that need to be performed in the next step. Figure 4.2 shows the axis-aligned culling region of the vehicle at two different locations in a point cloud and the corresponding isolated points. The corresponding algorithm for isolation of points using this method is described in Algorithm 2.
Figure 4.2: Subset of points isolated from original point clouds at two locations based on the position of vehicle and using axis-aligned culling region.

4.5.2 Point-Voxel Collision

Once the points in the axis-aligned culling region are isolated, we further check each of the points in $P_{AABB}$ for occupancy in each individual voxel in the voxel grid representation of the vehicle CAD model. For consistencies in the region around a voxel and clearance guarantees, we consider an inscribed sphere of a voxel for finding collisions with the point cloud. Since all the voxels in the grid are of consistent size and are cubical, the inscribed sphere’s diameter is the diagonal of the cubical voxel. The center of each voxel is the center of the corresponding spheres.
Algorithm 2: Axis-aligned culling region point isolation

Input: Point cloud, \( P_c \), axis-aligned bounding box \( V_{AABB} \)

Result: Subset of points, \( P_{AABB} \)

1. \( P_{AABB} \leftarrow \text{empty} \)
2. \( \text{bbox}_{\max} \leftarrow \max(V_{AABB}) \)
3. \( \text{bbox}_{\min} \leftarrow \min(V_{AABB}) \)
4. foreach Point \( P(x, y, z) \in P_c \) do
5. \quad if \( P(x, y, z) \leq \text{bbox}_{\max} \) then
6. \quad \quad if \( P(x, y, z) \geq \text{bbox}_{\min} \) then
7. \quad \quad \quad Add \( P(x, y, z) \rightarrow P_{AABB} \)
8. \quad \quad end
9. \quad end
10. end

To compute the collisions, each point in \( P_{AABB} \) is compared with every voxel in the grid to check for the occupancy of the point in the voxels. We compute the distance of each point from the center of the inscribed sphere of a voxel, and if the distance is less than the radius of the sphere (as shown in Figure 4.3), we mark the voxel as colliding or occupied using a scalar value in a separate data structure. Further, once a voxel is marked as colliding, we cull out the particular voxel from the consequent checks, thus reducing the number of computations required. This whole process is performed in parallel in the GPU with each thread computes the collision corresponding to each voxel and compares against all the points in \( P_{AABB} \) until a collision is found. The complete algorithm of the process is shown in Algorithm 3.

4.6 Clearance Analysis and Theoretical Guarantees

In this section, we provide an analysis of the clearance between the voxel representation of the vehicle with the point cloud of the environment while performing collision detection. Clearance \( D_{\text{clearance}} \) is given by the maximum distance between the colliding points and the center of the corresponding voxel. In our case, as we have used an in-
Figure 4.3: Example of a colliding point-voxel and a non-colliding point-voxel. $R_V$ is half the diagonal of the voxel. (a) shows a non-colliding point with a voxel as the distance of the point $P_1$ from voxel center is greater than $R_V$. (b) shows a colliding point with the voxel as distance from the point $P_2$ from the voxel center is less than $R_V$.

scribed sphere of a voxel to perform collision detection, $D_{\text{clearance}}$ remains uniform all over the voxel representation. This is computed as the radius of the inscribed sphere or half the length of the diagonal of the cubical voxel. Figure 4.4 shows the clearance distance of the humvee voxel representation with a voxel size of $a$. $D_{\text{clearance}}$ is computed using

1. $D_{\text{clearance}} = R_V$  
2. $= \frac{\sqrt{3}}{2}a$

The clearance distance relays the information that the collision of the vehicle with the environment point cloud is true up to a distance of $D_{\text{clearance}}$ from the vehicle’s boundary. However, for efficient navigation and better control over the collision detection, a user or designer might require more clearance to be conservative in making design decisions. A higher value of clearance (equivalent to factor or safety) is needed. Hence, we have developed a voxel-based Minkowski sum to augment the vehicle’s boundary to provide extra...
Algorithm 3: Parallel collision detection of point cloud and voxels

Input: Point cloud, \( P_{AABB} \), Voxel grid \( G \)
Result: Collision grid, \( G_{\text{collision}} \)

1. \( G_{\text{collision}} \leftarrow \text{empty} \); // Same size grid as \( G \)
2. \( \text{forall Voxel } V \in G \text{ in parallel do} \)
3. \( \text{bool collision } \leftarrow \text{false} \)
4. \( \text{Center}_V \leftarrow \text{Center}(V) \); // Voxel center
5. \( \text{foreach Point } P \in P_{AABB} \text{ do} \)
6. \( \text{if not Collision then} \)
7. \( \text{Dist}_P \leftarrow \text{Distance(}\text{Center}_V, P\text{)} \)
8. \( R_V \leftarrow \text{Diagonal}(V)/2 \)
9. \( \text{if Dist}_P \leq R_V \text{ then} \)
10. \( G_{\text{collision}}(V) \leftarrow 1.0 \)
11. \( \text{collision } \leftarrow \text{true} \)
12. \( \text{end} \)
13. \( \text{end} \)
14. \( \text{end} \)
15. \( \text{end} \)

clearance. Minkowski sum is used as a tool to offset (or thicken) the vehicle’s boundary voxels for this purpose.

4.6.1 Voxel based Minkowski Sums

Given two sets of vectors \( p, q \) representing two polygons \( P \) and \( Q \) respectively, the Minkowski sum of the two polygons in Euclidean space is given by:

\[
P \oplus Q = \{ p + q | p \in P, q \in Q \}.
\] (4.3)

In voxel space, this can be considered as a variant of convolution operation of a voxel grid representing \( Q \) (\( G_Q \)) over the voxel grid representing \( P \) (\( G_P \)). We perform the Minkowski sum in voxel space by convolving one grid with another and adding the overlapping voxel values (according to the grid indices) to get a final Minkowski voxel grid \( G_{\text{mink}} \) of size \( G_P + G_Q \). We then threshold \( G_{\text{mink}} \) for values greater than 0 to create the
Figure 4.4: Clearance distance of a boundary voxel of the Humvee voxel model. $R_V$ is the half diagonal length of the cubical voxel with size $a$.

voxel representation of the Minkowski sum. To preserve the sizes of the involved polygons or CAD models during the Minkowski sum operation, we voxelize them using the same voxel size. An example in 2D is shown in Figure 4.5.

To perform the Minkowski sum operation in voxel space, we first create an empty voxel grid for $G_{mink}$ with the number of voxels in $x$, $y$ and $z$ direction as the sum of the voxel grid sizes of $G_P$ and $G_Q$. Then we find a voxel $\hat{Q}$ on $G_Q$ (which is to be convolved) belonging to the polygon or CAD model (specifically, voxel with a scalar value of 1.0). This is done as we require to check $G_P$ and $G_Q$ for comparison and addition based on the corresponding index values of the voxel $\hat{Q}$. $\hat{Q}$ acts as the basis point of the polygon $Q$ while performing Minkowski operation. This is shown in detail in Figure 4.6. Further, this operation allows us to perform the Minkowski sum of non-convex polygons with the same algorithm and without any variation in complexity. We then copy the values of $G_P$ to an intermediate grid $G_{Pr}$ after offsetting the origin of the voxel grid with respect to the index of $\hat{Q}$. 
Convolution Operation with addition

Convolution Operation with addition

Final Minkowski sum grid, $G_{\text{mink}}$

Figure 4.5: 2D example of voxel based Minkowski sum. $P$ and $Q$ represent a triangle and a rectangle respectively. 2D voxel grid (pixels) of $Q$ is convolved over $P$ and the values are added with respect to overlapping voxels. $G_{\text{mink}}$ is the final minkowski sum grid obtained after the addition and thresholding of the values.

Once we isolate the voxel index of $\hat{Q}$, we proceed to perform the convolution operation of $G_{P'}$ and $G_Q$ by coinciding $\hat{Q}$ on each of the voxels of $G_{P'}$. If $\hat{Q}$ coincides with a filled voxel of $G_{P'}$ (or a voxel with value 1.0), we take the element-wise sum of $G_{P'}$ and $G_Q$ at that convolution location and store the respective voxel values in the Minkowski voxel grid after thresholding the sums greater than 0 to be 1.0. We repeat this operation for all the convolution steps until we get the complete Minkowski voxel grid. The convolution operation for calculating the Minkowski sum is parallelized on the GPU. The GPU-accelerated voxel-based Minkowski sum algorithm is described in Algorithm 4.

4.7 Results

In this section, we discuss the results of the collision detection algorithm and the voxel-based Minkowski sum method. We implement these algorithms in C++ programming language with the visualization and navigation being performed in Unreal Engine [6]. As the backend of the Unreal engine is in C++, it helped us to combine the implementation with the game engine easily. We use a large point cloud scan of a ship, as the enclosed
Figure 4.6: Comparison of polygons $P$ and $Q$ with respect to position of $\hat{Q}$ during convolution of voxel based minkowski sum. (a) shows a case when $\hat{Q}$ overlaps an inside voxel of $P$, hence index wise sum of $G_P$ and $G_Q$ are computed. (b) shows a case during convolution when $\hat{Q}$ does not overlap on $P$, hence no further sum is computed.

We import the point cloud into the game engine along with the humvee CAD model and visualize them. We then create a mesh in the game engine along the ship’s floor to represent the drive-able area so that the vehicle does not float in space. However, this does not affect the collision detection process.

After importing all the data into the game engine, we voxelize the vehicle CAD model into three different resolutions i.e., low, medium, and high, depending on the user’s requirement. For example, in this paper, we have used the vehicle’s voxel representation with a resolution of $64^3$ and $96^3$ as the medium and high resolutions, respectively. For these voxel resolutions, the respective voxel sizes are 3.7cm and 2.5cm which provides a clearance of 3.2cm and 2.1cm respectively.
Algorithm 4: Voxel based minkowski sum

Input: Voxel grids, $G_P$ and $G_Q$
Result: Minkowski sum grid, $G_{mink}$

1. $P_x, P_y, P_z \leftarrow$ voxel index of $G_P$
2. $Q_x, Q_y, Q_z \leftarrow$ voxel index of $G_Q$
3. $\hat{Q}_x, \hat{Q}_y, \hat{Q}_z \leftarrow$ null
4. foreach Voxel index, $(\text{indx}, \text{indy}, \text{indz}) \in G_Q$ do
   5. if $G_Q(\text{indx}, \text{indy}, \text{indz}) == 1.0$ then
      6. $\hat{Q}_x, \hat{Q}_y, \hat{Q}_z \leftarrow \text{indx}, \text{indy}, \text{indz}$
      7. break
   end
5. Set $G_P' \leftarrow 0$; // grid size of $(P_x + Q_x, P_y + Q_y, P_z, Q_z)$
6. Copy $G_P$ to $G_P'$ with an index offset of $\hat{Q}_x, \hat{Q}_y, \hat{Q}_z$
7. forall Voxel $V$ in parallel do
6. if $G_P'(P_x + Q_x, P_y + Q_y, P_z, Q_z) + G_Q(Q_x, Q_y, Q_z) \geq 2.0$ then
      7. foreach Voxel index, $(\text{indx}, \text{indy}, \text{indz}) \in G_Q$ do
         8. if $G_P'(P_x + \text{indx}, P_y + \text{indy}, P_z + \text{indz}) \geq 1.0$ then
            9. $G_{mink}(P_x + \text{indx}, P_y + \text{indy}, P_z + \text{indz}) = 1.0$
         end
      end
5. end
end

To visualize the vehicle’s voxels in the game engine, we store all the voxel center points along with other meta-data such as the number of voxels in the orthogonal directions, bounding box information, and voxel sizes. Then we import the point data into Unreal engine spawn voxels at the center points with the voxel size as given by the voxelization process. Figure 4.7 shows the rendering of the point cloud of the ship with two vehicles located at different positions.

To extend the vehicle’s clearance, we perform the GPU-accelerated voxel-based Minkowski sum on the vehicle voxel data with another voxel grid representing a cube of different sizes based on the clearance required. We make sure that the voxel sizes of the clearance grid are the same as the voxel size of the vehicle to preserve the scale infor-
Figure 4.7: Unreal Engine rendering of the ship point cloud and two vehicles at different locations with the voxel grid

For example, to create one layer of voxel around the boundary of the vehicle (clearance of $2 \times D_{\text{clearance}}$), we perform the Minkowski sum of the vehicle and another voxel grid of $2^3$ resolution with the same size voxels as the vehicle. The extra voxel resolution is used to compensate for the overlapping voxels while performing convolutions in the Minkowski sum algorithm. The resulting Minkowski sum voxel representations of the humvee are shown in Figure 4.8.

We then perform the collision detection of the vehicle’s obtained voxel grid with the environment point cloud. First, we get the vehicle’s current transformation matrix with respect to the point cloud based on its position and transform the vehicle in the collision module accordingly. Then we compute the collision of the point cloud with the voxel grid of the vehicle according to the steps explained in Section 4.5. The collision detection also stores a replica voxel grid of the vehicle with the collided voxels having a value of 1.0. We render this back in the game engine, with only the collided voxels overlapped on the CAD model of the vehicle. Figure 4.9 and Figure 4.10 shows the collision of the vehicle in two separate locations of the point cloud.
Figure 4.8: Volumetric rendering of a Humvee CAD model with Minkowski sum operation. (a) shows isometric, front and side views of original voxel model. (b) shows minkowski sum of the original voxel model with another voxel grid of resolution $2^3$ (1 extra voxel). (c) shows the minkowski sum with a voxel grid of resolution $4^3$ (3 extra voxels).

4.8 Conclusion

In this paper, we have developed a GPU-accelerated collision detection framework for navigation of vehicles in point cloud representations of enclosed spaces using voxels. The collision is computed using the voxel representation of the vehicle with the point cloud, thus eliminating the issues of collision with a tessellated model. We also provide a clearance analysis of the collision with theoretical guarantees. For better control over the clearance and collision detection, we developed a GPU-accelerated voxel-based Minkowski sum algorithm to offset the vehicle’s boundary voxels as per user requirement. We also implemented the complete framework in a game engine with appropriate rendering to guide the user interactively in making design decisions. Future work involves adding interactive clearance visualizations and path navigation of the vehicle in enclosed spaces using the collision detection algorithm.
Figure 4.9: Unreal Engine rendering of the collision of humvee voxels with the point cloud of the ship. The red voxels show the exact voxel locations where there is a collision detected.
Figure 4.10: Collision detection of voxels in point clouds at two different locations.

4.9 References


CHAPTER 5. SPATIOTEMPORALLY CONSTRAINED ACTION
SPACE ATTACKS ON DEEP REINFORCEMENT LEARNING AGENTS

A paper accepted by Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence
Xian Yeow Lee, Sambit Ghadai, Kailiang Tan, Chinmay Hegde and Soumik Sarkar

5.1 Abstract

Robustness of Deep Reinforcement Learning (DRL) algorithms towards adversarial attacks in real world applications such as those deployed in cyber-physical systems (CPS) are of increasing concern. Numerous studies have investigated the mechanisms of attacks on the RL agent’s state space. Nonetheless, attacks on the RL agent’s action space (corresponding to actuators in engineering systems) are equally perverse, but such attacks are relatively less studied in the ML literature. In this work, we first frame the problem as an optimization problem of minimizing the cumulative reward of an RL agent with decoupled constraints as the budget of attack. We propose the white-box Myopic Action Space (MAS) attack algorithm that distributes the attacks across the action space dimensions. Next, we reformulate the optimization problem above with the same objective function, but with a temporally coupled constraint on the attack budget to take into account the approximated dynamics of the agent. This leads to the white-box Look-ahead Action
Space (LAS) attack algorithm that distributes the attacks across the action and temporal dimensions. Our results showed that using the same amount of resources, the LAS attack deteriorates the agent’s performance significantly more than the MAS attack. This reveals the possibility that with limited resource, an adversary can utilize the agent’s dynamics to malevolently craft attacks that causes the agent to fail. Additionally, we leverage these attack strategies as a possible tool to gain insights on the potential vulnerabilities of DRL agents.

5.2 Introduction

The spectrum of Reinforcement Learning (RL) applications ranges from engineering design and control [18, 29] to business [12] and creative design [23]. As RL-based frameworks are increasingly deployed in real-world, it is imperative that the safety and reliability of these frameworks are well understood. While any adversarial infiltration of these systems can be costly, the safety of DRL systems deployed in cyber-physical systems (CPS) such as industrial robotic applications and self-driving vehicles are especially safety and life-critical.

A root cause of these safety concerns is that in certain applications, the inputs to an RL system can be accessed and modified adversarially to cause the RL agent to take sub-optimal (or even harmful) actions. This is especially true when deep neural networks (DNNs) are used as key components (e.g., to represent policies) of RL agents. Recently, a wealth of results in the ML literature demonstrated that DNNs can be fooled to misclassify images by perturbing the input by an imperceptible amount [9] or by introducing specific natural looking attributes [15]. Such adversarial perturbations have also demonstrated the impacts of attacks on an RL agent’s state space as shown by [13].
Besides perturbing the RL agent’s state space, it is also important to consider adversarial attacks on the agent’s action space, which in engineering systems, represents physically manipulable actuators. We note that (model-based) actuator attacks have been studied in the cyber-physical security community, including vulnerability of continuous systems to discrete time attacks [16]; theoretical characteristics of undetectable actuator attacks [1]; and “defense” schemes that re-stabilizes a system when under actuation attacks [14]. However, the issue of adversarial attacks on a RL agent’s action space has relatively been ignored in the DRL literature. In this work, we present a suite of novel attack strategies on a RL agent’s action space.

**Our contributions:**

1. We formulate a white-box Myopic Action Space (MAS) attack strategy as an optimization problem with decoupled constraints.

2. We extend the formulation above by coupling constraints to compute a non-myopic attack that is derived from the agent’s state-action dynamics and develop a white-box Look-ahead Action Space (LAS) attack strategy. Empirically, we show that LAS crafts a stronger attack than MAS using the same budget.

3. We illustrate how these attack strategies can be used to understand a RL agent’s vulnerabilities.

4. We present analysis to show that our proposed attack algorithms leveraging projected gradient descent on the surrogate reward function (represented by the trained RL agent model) converges to the same effect of applying projected gradient descent on the true reward function.
Table 5.1: Landscape of adversarial attack strategies on RL agents. First column denotes if the attack takes into account the dynamics of the agent. Second column shows the method of computing the attacks; $O$ denotes an optimization-based method and $M$ denotes a model-based method where the parameters of a model needs to be learned. Last column represents if the attacks are mounted on agent’s state space (S) or action space (A).

<table>
<thead>
<tr>
<th>Method</th>
<th>Includes Dynamics</th>
<th>Method</th>
<th>Space of Attack</th>
</tr>
</thead>
<tbody>
<tr>
<td>FGSM on Policies [13]</td>
<td>X</td>
<td>O</td>
<td>S</td>
</tr>
<tr>
<td>ATN [32]</td>
<td>X</td>
<td>M</td>
<td>S</td>
</tr>
<tr>
<td>Gradient based Adversarial Attack [22]</td>
<td>X</td>
<td>O</td>
<td>S</td>
</tr>
<tr>
<td>Policy Induction Attacks [3]</td>
<td>X</td>
<td>O</td>
<td>S</td>
</tr>
<tr>
<td>NR-MDP [30]</td>
<td>X</td>
<td>M</td>
<td>A</td>
</tr>
<tr>
<td>Myopic Action Space (MAS)</td>
<td>X</td>
<td>O</td>
<td>A</td>
</tr>
<tr>
<td>Look-ahead Action Space (LAS)</td>
<td>✓</td>
<td>O</td>
<td>A</td>
</tr>
</tbody>
</table>

5.3 Related Works

Due to the large amount of recent progress in the area of adversarial machine learning, we only focus on reviewing the most recent attack and defense mechanisms proposed for DRL models. Table 5.1 presents the primary landscape of this area of research to contextualize our work.

5.3.1 Adversarial Attacks on RL Agent

Several studies of adversarial attacks on DRL systems have been conducted recently. [13] extended the idea of FGSM attacks in deep learning to RL agent’s policies to degrade the performance of a trained RL agent. Furthermore, [3] showed that these attacks on the agent’s state space are transferable to other agents. Additionally, [32] proposed attaching an Adversarial Transformer Network (ATN) to the RL agent to learn perturbations that will deceive the RL agent to pursue an adversarial reward. While the attack strategies mentioned above are effective, they do not consider the dynamics of the agent. One ex-
ception is the work by [19] that proposed two attack strategies. One strategy was to attack the agent when the difference in probability/value of the best and worst action crosses a certain threshold. The other strategy was to combine a video prediction model that predicts future states and a sampling-based action planning scheme to craft adversarial inputs to lead the agent to an adversarial goal, which might not be scalable. Other studies of adversarial attacks on the specific application of DRL for path-finding have also been conducted by [35] and [2], which results in the RL agent failing to find a path to the goal or planning a path that is more costly.

5.3.2 Robustification of RL Agents

As successful attack strategies are being developed for RL models, various works on training RL agents to be robust against attacks have also been conducted. [22] proposed that a more severe attack can be engineered by increasing the probability of the worst action rather than decreasing the probability of the best action. They showed that the robustness of an RL agent can be improved by training the agent using these adversarial examples. More recently, [30] presented a method to robustify RL agent’s policy towards action space perturbations by formulating the problem as a zero-sum Markov game. In their formulation, a separate nominal and adversary policy are trained simultaneously with a critic network being updated over the mixture of both policies to improve both adversarial and nominal policies. Meanwhile, [11] proposed a method to detect and mitigate attacks by employing a hierarchical learning framework with multiple sub-policies. They showed that the framework reduces agent’s bias to maintain high nominal rewards in the absence of adversaries. We note that other methods to defend against adversarial attacks exist, such as studies done by [31] and [26]. These works are done mainly in the context of a DNN but may be extendable to DRL agents that employs DNN as policies, however discussing these works in detail goes beyond the scope of this work.
5.4 Mathematical Formulation

5.4.1 Preliminaries

We focus exclusively on model-free RL approaches. Below, let $s_t$ and $a_t$ denote the (continuous, possibly high-dimensional) vector variables denoting state and action, respectively, at time $t$. We assume a state evolution function, $s_{t+1} = E(s_t, a_t)$ and let $R(s_t, a_t)$ denote the reward signal the agent receives for taking the action $a_t$, given $s_t$. The goal of the RL agent is to choose actions that maximizes the cumulative reward, $\sum_t R(s_t, a_t)$, given access to the trajectory, $\tau$, comprising all past states and actions. In value-based methods, the RL agent determines action at each time step by finding an intermediate quantity called the value function that satisfies the recursive Bellman Equations. One example of such method is Q-learning [34] where the agent discovers the Q-function, defined recursively as:

$$Q_t(s_t, a_t) = R(s_t, a_t) + \max_{a'} Q_{t+1}(E(s_t, a_t), a').$$

The optimal action (or “policy”) at each time step is to deterministically select the action that maximizes this Q-function conditioned on the observed state, i.e.,

$$a_t^* = \arg \max_a Q(s_t, a).$$

In DRL, the Q-function in the above formulation is approximated via a parametric neural network $\Theta$; methods to train these networks include Deep Q-networks [21].

In policy-based methods such as policy gradients [28], the RL agent directly maps trajectories to policies. In contrast with Q-learning, the selected action is sampled from the policy parameterized by a probability distribution, $\pi = P(a|s, \Theta)$, such that the expected rewards (with expectations taken over $\pi$) are maximized:

$$\pi^* = \arg \max_{\pi} E[R(\tau)], \ a_t^* \sim \pi^*. $$
In DRL, the optimal policy $\pi$ is the output of a parametric neural network $\Theta$, and actions at each time step are sampled; methods to train this neural network include proximal policy optimization (PPO) [25].

5.4.2 Threat Model

Our goal is to identify adversarial vulnerabilities in RL agents in a principled manner. To this end, we define a formal threat model, where we assume the adversary possesses the following capabilities:

1. **Access to RL agent’s action stream.** The attacker can directly perturb the agent’s nominal action adversarially (under reasonable bounds, elaborated below). The nominal agent is also assumed to be a closed-loop system and have no active defense mechanisms.

2. **Access to RL agent’s training environment.** This is required to perform forward simulations to design an optimal sequence of perturbations (elaborated below).

3. **Knowledge of trained RL agent’s DNN.** This is needed to understand how the RL agent acts under nominal conditions, and to compute gradients. In the adversarial ML literature, this assumption is commonly made under the umbrella of white-box attacks.

In the context of the above assumptions, the goal of the attacker is to choose a (bounded) action space perturbation that minimizes long-term discounted rewards. Based on how the attacker chooses to perturb actions, we define and construct two types of optimization-based attacks. We note that alternative approaches, such as training another RL agent to learn a sequence of attacks, is also plausible. However, an optimization-based approach is computationally more tractable to generate on-the-fly attacks for a target
agent compared to training another RL agent (especially for high-dimensional continuous action spaces considered here) to generate attacks. Therefore, we restrict our focus on optimization-based approaches in this paper.

5.4.3 Myopic Action-Space (MAS) Attack Model

We first consider the case where the attacker is myopic, i.e., at each time step, they design perturbations in a greedy manner without regards to future considerations. Formally, let $\delta_t$ be the action space perturbation (to be determined) and $b$ be a budget constraint on the magnitude of each $\delta_t$. At each time step $t$, the attacker designs $\delta_t$ such that the anticipated future reward is minimized

$$
\min_{\delta_t} R_{\text{adv}}(\delta_t) = R(s_t, a_t + \delta_t) + \sum_{j=t+1}^{T} R(s_j, a_j)
$$

subject to:

$$
\|\delta_t\|_p \leq b,
$$

$$
E(s_{j+1}) = E(s_j, a_j),
$$

$$
a_j = \Theta(s_j) \text{ (for } j = t, \ldots, T),
$$

where $\|\cdot\|_p$ denotes the $\ell_p$-norm for some $p \geq 1$. Observe that while the attacker ostensibly solves separate (decoupled) problems at each time, the states themselves are not independent since given any trajectory, $s_{j+1} = E(s_j, a_j)$, where $E(s_j, a_j)$ is the transition of the environment based on $s_j$ and $a_j$. Therefore, $R$ is implicitly coupled through time since it depends heavily on the evolution of state trajectories rather than individual state visitations. Hence, the adversary perturbations solved above are strictly myopic and we consider this a static attack on the agent’s action space.

\footnote{Physically, the budget may reflect a real physical constraint, such as the energy requirements to influence an actuation, or it may be a reflection on the degree of imperceptibility of the attack.}
5.4.4 Look-ahead Action Space (LAS) Attack Model

Next, we consider the case where the attacker is able to look ahead and chooses a designed sequence of future perturbations. Using the same notation as above, let 
\[ \sum_{j=t}^{t+H} R(s_j, a_j + \delta_j) \] denote the sum of rewards until a horizon parameter \( H \), and let 
\[ \sum_{j=t+H+1}^{T} R(s_j, a_j) \] be the future sum of rewards from time \( j = t + H + 1 \). Additionally, we consider the (concatenated) matrix \( \Delta = [\delta_t, \delta_{t+1} \ldots \delta_{t+H}] \) and \( B \) denote a budget parameter. The attacker solves the optimization problem:

\[
\min_{\Delta} R_{\text{adv}}(\Delta) = \sum_{j=t}^{t+H} R(s_j, a_j + \delta_j) + \sum_{j=t+H+1}^{T} R(s_j, a_j)
\]

subject to: 
\[ \| \Delta \|_{p,q} \leq B, \Delta = [\delta_t, \delta_{t+1}, \ldots, \delta_{H}], \] 

\[ s_{j+1} = E(s_j, a_j), \]

\[ a_j = \Theta(s_j) \]

where \( \| \cdot \|_{p,q} \) denotes the \( \ell_{p,q} \)-norm [4]. By coupling the objective function and constraints through the temporal dimension, the solution to the optimization problem above is then forced to take the dynamics of the agent into account in an explicit manner.

5.5 Proposed Algorithms

In this section, we present two attack algorithms based on the optimization formulations presented in previous section.

5.5.1 Algorithm for Mounting MAS Attacks

We observe that (5.1) is a nonlinear constrained optimization problem; therefore, an immediate approach to solve it is via projected gradient descent (PGD). Specifically, let \( S \) denote the \( \ell_p \) ball of radius \( b \) in the action space. We compute the gradient of the
adversarial reward, $\nabla R_{\text{adv}}$ w.r.t. the action space variables and obtain the *unconstrained* adversarial action $\hat{a}_{t+\frac{1}{2}}$ using gradient descent with step size $\eta$. Next, we calculate the *unconstrained* perturbation $\delta_t$ and project it onto $S$ to get $\delta'_t$:

\[
\begin{align*}
\hat{a}_{t+\frac{1}{2}} &= a_t - \eta \nabla R_{\text{adv}}(s_t, \hat{a}_t), \\
\delta_t &= \hat{a}_{t+\frac{1}{2}} - a_t, \\
\delta'_t &= P_S(\delta_t).
\end{align*}
\]

Here, $a_t$ represents the nominal action. We note that this approach resembles the fast gradient-sign method (FGSM) [9], although we compute standard gradients here. As a variation, we can compute multiple steps of gradient descent w.r.t. the action variable prior to projection; this is analogous to the basic iterative method (or iterative FGSM) [17]. The MAS attack algorithm is shown in the supplementary material.

We note that in DRL approaches, only a *noisy proxy* of the true reward function is available: In value-based methods, we utilize the learned Q-function (for example, a DQN) as an approximate of the true reward function, while in policy-iteration methods, we use the probability density function returned by the optimal policy as a proxy of the reward, under the assumption that actions with high probability induce a high expected reward. Since DQN selects the action based on the argmax of Q-values and policy iteration samples the action with highest probability, the Q-values/action-probability remains a useful proxy for the reward in our attack formulation. Therefore, our proposed MAS attack is technically a version of *noisy projected gradient descent* on the policy evaluation of the nominal agent. We elaborate on this further in the theoretical analysis section.

### 5.5.2 Algorithm for Mounting LAS Attacks

The previous algorithm is myopic and can be interpreted as a purely *spatial* attack. In this section, we propose a *spatiotemporal* attack algorithm by solving Eq. (5.2) over
Algorithm 5: Look-ahead Action Space (LAS) Attack

1. Initialize nominal and adversary environments $E_{nom}$, $E_{adv}$ with same random seed
2. Initialize nominal agent $\pi_{nom}$ weights, $\theta$
3. Initialize budget $B$, adversary action buffer $A_{adv}$, horizon $H$
4. while $t \leq T$ do
5. \hspace{1em} Reset $A_{adv}$
6. \hspace{1em} if $H = 0$ then
7. \hspace{2em} Reset $H$ and $B$
8. \hspace{1em} while $k \leq H$ do
9. \hspace{2em} Compute gradient of surrogate reward $\nabla R_{adv}$
10. \hspace{2em} Compute adversarial action $\hat{a}_{t+\frac{1}{2}k}$ using $\nabla R_{adv}$
11. \hspace{2em} Compute $\delta_{t,k} = \hat{a}_{t+\frac{1}{2}k} - a_{t,k}$
12. \hspace{2em} Append $\delta_{t,k}$ to $A_{adv}$
13. \hspace{2em} Step through $E_{adv}$ with $a_{t,k}$ to get next state
14. \hspace{1em} end
15. Compute $||\delta_{t,k}||_{\ell_p}$ for each element in $A_{adv}$
16. Project sequence of $||\delta_{t,k}||_{\ell_p}$ in $A_{adv}$ on to ball of size $B$ to obtain look-ahead sequence of budgets $[b_{t,k}, b_{t,k+1} \ldots b_{t,k+H}]$
17. Project each $\delta_{t,k}$ in $A_{adv}$ on to look-ahead sequence of budgets computed in the previous step to get sequence $[\delta'_{t,k}, \delta'_{t,k+1} \ldots \delta'_{t,k+H}]$
18. Compute projected adversarial action $\hat{a}_t = a_t + \delta'_{t,k}$
19. Step through $E_{nom}$ with $\hat{a}_t$
20. $B \leftarrow \max(0, B - \delta'_{t,k})$
21. $H \leftarrow H - 1$
22. end

a given time window $H$. Due to the coupling of constraints in time, this approach is more involved. We initialize a copy of both the nominal agent and environment, called the adversary and adversarial environment respectively. At time $t$, we sample a virtual roll-out trajectory up until a certain horizon $t + H$ using the pair of adversarial agent and environment. At each time step $k$ of the virtual roll-out, we compute action space perturbations $\delta_{t,k}$ by taking (possibly multiple) gradient updates. Next, we compute the norms of each $\delta_{t,k}$ and project the sequence of norms back onto an $\ell_q$-ball of radius $B$. The resulting projected norms at each time point now represents the individual budgets, $b_k$,
of the spatial dimension at each time step. Finally, we project the original $\delta_{t,k}$ obtained in the previous step onto the $\ell_p$-balls of radii $b_k$, respectively to get the final perturbations $\delta'_{t,k}$. We note that to perform virtual roll-outs at every time step $t$, the state of the $E_{adv}$ has to be the same as the state of $E_{nom}$ at $t$. To accomplish this, we saved the history of all previous actions to re-compute the state of the $E_{adv}$ at time $t$ from $t = 0$. While this current implementation may be time-consuming, we believe that this problem can be avoided by giving the adversary direct access to the current state of the nominal agent through platform API-level modifications; or explicit observations (in real-life problems).

In subsequent time steps, the procedure above is repeated with a reduced budget of $B - \sum_{t=0}^{t} \delta'_t$ and reduced horizon $H - t$ until $H$ reaches zero. The horizon $H$ is then reset again for planning a new spatiotemporal attack. An alternative formulation could also be shifting the window without reducing its length until the adversary decides to stop the attack. However, we consider the first formulation such that we can compare the performance of LAS with MAS for an equal overall budget. This technique of re-planning the $\delta'_t$ at every step while shifting the window of $H$ is similar to the concept of receding horizons regularly used in optimal control [20]. It is evident that using this form of dynamic re-planning mitigates the planning error that occurs when the actual and simulated state trajectories diverge due to error accumulation [24]. Hence, we perform this re-planning at every $t$ to account for this deviation. The pseudocode is provided in Alg. 5.

5.5.3 Theoretical Analysis

We can show that projected gradient descent on the surrogate reward function (modeled by the RL agent network) to generate both MAS and LAS attacks provably converges; this can be accomplished since gradient descent on a surrogate function is akin to a noisy gradient descent on the true adversarial reward.

\footnote{Intuitively, these steps represent the allocation of overall budget $B$ across different time steps.}
Figure 5.1: Visual comparison of MAS and LAS. In MAS, each $\delta_t$ is computed via multi-step gradient descent w.r.t. expected rewards for the current step. In LAS, each $\delta'_{t,k}$ is computed w.r.t. the dynamics of the agent with receding horizon. An adversarial agent & environment is used to compute LAS for each step. Projection is applied to each $\delta_t$ in the temporal domain. The final perturbed action is obtained by adding the first $\delta'_{t,k}$ to the nominal action. This is done until the end of the attack window, i.e., $H - t = 0$.

As described in previous sections, our MAS/LAS algorithms are motivated in terms of the adversarial reward $R_{adv}$. However, if we use either DQN or policy gradient networks, we do not have direct access to the reward function, but only its noisy proxy, defined via a neural network. Therefore, we need to argue that performing (projected) gradient descent using this proxy loss function is a sound procedure. To do this, we appeal to a recent result by [8], who prove convergence of noisy gradient descent approximately converges to a local minimum. More precisely, consider a general constrained nonlinear
optimization problem:

$$\min f(x)$$

s.t. $$c(x) = 0,$$

where $$c$$ is an arbitrary (differentiable, possibly vector-valued) function encoding the constraints. Define $$S = \{x|c(x) = 0\}$$ define the constraint set. We attempt to minimize the objective function via noisy (projected) gradient updates:

$$x_{t+1/2} = x_t - \eta \nabla f(x_t) + \xi_t,$$

$$x_{t+1} = P_S(x_{t+1/2}).$$

**Theorem 1.** (Convergence of noisy projected gradients.) Assume that the noise terms $$\{\xi_t\}$$ are i.i.d., satisfying $$E[\xi] = 0, E[\xi \xi^T] = \sigma^2 I, \|\xi\| \leq O(1)$$ almost surely. Assume that both the constraint function $$c()$$ and the objective function $$f(\cdot)$$ is $$\beta$$-smooth, $$L$$-Lipschitz, and possesses $$\rho_L$$-Lipschitz Hessian. Assume further that the objective function $$f$$ is $$B$$-bounded. Then, there exists a learning rate $$\eta = O(1)$$ such that with high probability, in polylog $$\left(\frac{1}{\eta^2}\right)$$ iterations, noisy projected gradient descent converges to a point $$\hat{x}$$ that is polylog $$(\sqrt{\eta})$$-close to some local minimum of $$f$$.

In our case, $$f$$ and $$\xi$$ depends on the structure of the RL agent’s neural network. (Smoothness assumptions of $$f$$ can perhaps be justified by limiting the architecture of the network, but the iid-ness assumption on $$\xi$$ is hard to verify). As such, it is difficult to ascertain whether the assumptions of the above theorem are satisfied in specific cases. Nonetheless, an interesting future theoretical direction is to understand Lipschitz-ness properties of specific families of DQN/policy gradient agents.

We defer further analysis of the double projection step onto mixed-norm balls used in our proposed LAS algorithms to the supplementary material.
5.6 Experimental Results & Discussion

To demonstrate the effectiveness and versatility of our methods, we implemented them on RL agents with continuous action environments from OpenAI’s gym [5] as they reflect the type of action space in most practical applications. For policy-based methods, we trained a nominal agent using the PPO algorithm and a DoubleDQN (DDQN) agent [33] for value-based methods. Additionally, we utilize Normalized Advantage Functions [10] to convert the discrete nature of DDQN’s output to continuous action space. For succinctness, we present the results of the attack strategies only on PPO agent for the Lunar-Lander environment. As a baseline, we implemented a random action space attack, where a random perturbation bounded by the same budget \( b \) is applied to the agent’s action space at every step. For MAS attacks, we implemented two different spatial projection schemes, \( \ell_1 \) projection based on [7] that represents a sparser distribution and \( \ell_2 \) projection that represents a denser distribution of attacks. For LAS attacks, all combinations of spatial and temporal projection for \( \ell_1 \) and \( \ell_2 \) were implemented.

5.6.1 Comparison of MAS and LAS Attacks

Figure 5.2 shows distributions of cumulative rewards obtained by the PPO agent across ten episodes in a Lunar Lander environment, with each subplot representing different combinations of budget, \( B \) and horizon, \( H \). Top three subplots show experiments with a \( H \) value of 5 time steps and \( b \) value of 3, 4, and 5 from left to right respectively. Bottom row of figures show a similar set of experiments but with a \( H \) value of 10. For a direct comparison between MAS and LAS attacks with equivalent budgets across time, we have

\footnote{Python codes and links to supplementary are available at \url{https://github.com/xylee95/Spatiotemporal-Attack-On-Deep-RL-Agents}}

\footnote{The only difference in implementation of policy vs value-based methods is that in policy methods, we take analytical gradients of a distribution to compute the attacks (e.g., in line 10 of Algorithm 5) while for value-based methods, we randomly sample adversarial actions to compute numerical gradients.}
assigned the corresponding MAS budget values as $b = B/H$. This assumes that the total budget $B$ is allocated uniformly across every time step for a given $H$, while LAS has the flexibility to allocate the attack budget non-uniformly in the same interval, conditioned on the dynamics of the agent.

We note that keeping $H$ constant while increasing $B$ provides both MAS and LAS with a higher budget to inject $\delta_i$ to the nominal actions. We observe that with a low budget of 3 (Figure 5.2a), only LAS is successful in attacking the RL agent, as seen by the corresponding decrease in rewards. With a higher budget of 5 (Figure 5.2c), MAS...
has a more apparent effect on the performance of the RL agent while LAS reduces the performance of the agent severely.

With $B$ constant, increasing $H$ allows the allocated $B$ to be distributed along the increased time horizon. In other words, LAS virtually looks-ahead further into the future. In the most naive case, a longer horizon dilutes the severity of each $\delta_t$ in compared to shorter horizons. By comparing similar budget values of different horizons (i.e. horizons 5 and 10 for budget 3, Figure 5.2a and Figure 5.2d respectively), attacks for $H = 10$ are generally less severe than their $H = 5$ counterparts. For all $B$ and $H$ combinations, we observe that MAS attacks are generally less effective compared to LAS. We note that this is a critical result of the study as most literature on static attacks have shown that the attacks can be ineffective below a certain budget. Here, we demonstrate that while MAS attacks can seemingly look ineffective for a given budget, a stronger and more effective attack can essentially be crafted using LAS with the same budget.

In the following sections, we further study the difference between MAS and LAS as well as demonstrate how the attacks can be utilized to understand the vulnerabilities of the agent in different environments.

5.6.2 Action Dimension Decomposition of LAS Attacks

Figure 5.3 shows action dimension decomposition of LAS attacks. The example shown in Figure 5.3 is the result of $\ell_2$ projection in action space with $\ell_2$ projection in time. From Fig. 5.3a, we observe that through all the episodes of LAS attacks, one of the action dimension (i.e., Up - Down direction of lunar lander) is consistently perturbed more, i.e., accumulates more attack, than Left-Right direction.

Figure 5.3b shows a detailed view of action dimension attacks for an episode (Episode 1). It is evident from the figure that the Up-Down actions of the lunar lander are more prone to attacks throughout the episode than Left-Right actions. Additionally, Left-Right
Figure 5.3: Time vs Attack magnitude along action dimension for LAS attacks with $B = 4, H = 5$ in Lunar Lander environment with PPO RL agent. (a) Variation of attack magnitude along Up-Down and Left-Right action dimensions through different episodes. In all episodes except episode 2, Up-Down action is more heavily attacked than Left-Right. (b) Variation of attack magnitude through time for episode 1 of (a). After 270 steps, the agent is not attacked in the Left-Right dimension, but heavily attacked in Up-Down directions. (c) Actual rendering of Lunar Lander environment for episode 1 of (a) corresponding to (b). Frame 1-5 are strictly increasing time steps showing trajectory of the RL agent controlling the lunar lander.

action attacks are restricted after certain time steps and only the Up-Down actions are attacked further. Figure 5.3c further corroborates the observation in the Lunar Lander environment. As the episode progresses in Figure 5.3c, the lunar lander initially lands on the ground in frame 3, but lifts up and hovers until the episode ends in frame 5. This observation supports the fact that the proposed attacks are effective in perturbing the action dimensions in an optimal manner; as in this case, perturbing the lunar lander in the horizontal direction will not further decrease rewards. On the other hand, hovering the lunar lander will cause the agent to use more fuel, which consequently decreases the total reward. From these studies, it can be concluded that LAS attacks (correlated with
projections of actions in time) can clearly isolate vulnerable action dimension(s) of the RL agent to mount a successful attack.

5.6.3 Ablation Study of Horizon and Budget

Lastly, we performed multiple ablation studies to compare the effectiveness of LAS and MAS attacks. While we have observed that LAS attacks are generally stronger than MAS, we hypothesize that there will be an upper limit to LAS’s advantage as allowable budget increases. We take the difference of each attack’s reduction in rewards (i.e. attack - nominal) and visualize how much rewards LAS reduces as compared to MAS under different conditions of $B$ and $H$. In case of PPO in Lunar Lander, we observe that reduction in rewards of LAS vs MAS becomes less drastic as budget increases, hence showing that LAS has diminishing returns as both MAS and LAS saturates at higher budgets.

5.7 Conclusion & Future Work

In this study, we present two novel attack strategies on an RL agent’s action space; a myopic attack (MAS) and a non-myopic attack (LAS). The results show that LAS attacks, that were crafted with explicit use of the agent’s dynamics information, are more powerful than MAS attacks. Additionally, we observed that applying LAS attacks on RL agents reveals the possible vulnerable actuators of an agent, as seen by the non-uniform distribution of attacks on certain action dimensions. This can be leveraged as a tool to identify the vulnerabilities and plan a mitigation strategy under similar attacks. Possible future works include extending the concept of LAS attacks to state space attacks where the agent’s observations are perturbed instead of the agent’s actions while taking into account the dynamics of the agent. Additionally, while we did not focus on the imperceptibility and deployment aspects of the proposed attacks in this study, defining a proper metric in
terms of detectability in action space and optimizing the budget to remain undetected for different environments will be a future research direction.

5.8 Appendix: Supplementary Materials

5.8.1 Pseudocode of MAS Attack

5.8.2 Analysis

5.8.2.1 Projections onto Mixed-norm Balls

The Look-ahead Action Space (LAS) Attack Model described above requires projecting onto the mixed-norm $\ell_{p,q}$-ball of radius $B$ in a vector space. Below, we show how to provably compute such projections in a computationally efficient manner. For a more complete treatment, we refer to [27]. Recall the definition of the $(p,q)$-norm. Let $X \in \mathbb{R}^{m \times n}$ be partitioned into sub-vectors $x_i$, $i \in [n]$ of length $m$. Then,

$$\|X\|_{p,q} := \left( \sum_{i=1}^{n} \|x_i\|_q^p \right)^{1/p}.$$  

Due to scale invariance of norms, we can assume $B = 1$. We consider the following special cases:

**Algorithm 6: Myopic Action Space (MAS) Attack**

1. Initialize nominal environment, $E_{nom}$, nominal agent $\pi_{nom}$ with weights, $\theta$
2. Initialize budget $b$
3. **while** $t \leq T$ **do**
   4. Compute gradient of surrogate reward $\nabla R_{adv}$
   5. Compute adversarial action $\hat{a}_{t+\frac{1}{2}}$ using $\nabla R_{adv}$
   6. Compute $\delta_t = \hat{a}_{t+\frac{1}{2}} - a_t$, project $\delta_t$ onto ball of size $b$ to get $\delta'_t$
   7. Compute projected adversarial action $\hat{a}_t = a_t + \delta'_t$
   8. Step through $E_{nom}$ with $\hat{a}_t$ to get next state
4. **end**
1. \( p = 1, q = 1 \): this reduces to the case of the ordinary \( \ell_1 \)-norm in \( \mathbb{R}^{mn} \). Projection onto the unit \( \ell_1 \)-ball can be achieved via *soft-thresholding* every entry in \( X \):

\[
P_S(X_{ij}) = \text{sign}(X_{ij}) \cdot (|X_{ij}| - \lambda)_+,
\]

where \( \lambda > 0 \) is a KKT parameter that can be discovered by a simple sorting the (absolute) values of \( X \). See [6].

2. \( p = 2, q = 2 \): this reduces to the case of the isotropic \( \ell_2 \)-norm in \( \mathbb{R}^{mn} \). Projection onto the unit \( \ell_2 \)-ball can be achieved by simple normalization:

\[
P_S(X_{ij}) = X_{ij} / \|X\|_{2,2}.
\]

3. \( p = 1, q = 2 \): this is a “hybrid” combination of the above two cases, and corresponds to the procedure that we use in mounting our LAS attack. Projection onto this ball can be achieved by a three-step method. First, we compute the \( n \)-dimensional vector, \( v \), of column-wise \( \ell_2 \)-norms. Then, we project \( v \) onto the unit \( \ell_1 \)-ball; essentially, this enables us to “distribute” the (unit) budget across columns. Since \( \ell_1 \)-projection is achieved via soft-thresholding, a number of coordinates of this vector are zeroed out, and others undergo a shrinkage. Call this (sparsified) projected vector \( v_p \). Finally, we perform an \( \ell_2 \) projection, i.e., we scale each column of \( X \) by dividing by its norm and multiplying by the entries of \( v_p \):

\[
P_S(X_{ij}) = X_{ij} / \|x_j\|_2 \cdot v_p(i).
\]

5.8.3 Comparison of Attacks Mounted on PPO Agent in Bipedal-Walker Environment

The results in Figure 5.4 depicts the comparison between the MAS and LAS attacks applied on a PPO agent in the Bipedal-Walker environment. A similar trend is observed where LAS attacks are generally more severe than MAS attacks. We acknowledge that in
this environment, MAS attacks are sometimes effective in reducing the rewards as well. However, this can be attributed to the Bipedal Walker having more dimensions (4 dimensions) in terms of its action space in compared to the Lunar-Lander (2 dimensions) environment. In addition, the actions of the Bipedal Walker is also highly coupled, in compared to the actions of the Lunar Lander. Hence, the agent for Bipedal-Walker is more sensitive towards perturbations, which explains the increase efficacy of MAS attacks.

5.8.4 Comparison of Attacks on DDQN Agent in Lunar Lander and Bipedal-Walker Environments

In Figures 5.5 and 5.6, we present additional results on the efficacy of different attack strategies for a DoubleDQN agent trained in the Lunar Lander and Bipedal Walker
Figure 5.5: DDQN Lunar Lander box plots showing average cumulative reward across 10 episodes for each attack method. Plots (a), (b) and (c) are attacked with a horizon of 5 time steps with budget value of 3, 4, and 5 respectively. (d), (e), and (f) are attacked with a horizon value of 10 time steps with budget value of 3, 4, and 5 respectively. Given the same horizon and budget, it is evident LAS attacks are more severe than MAS attacks, which in turn are generally more effective than random attacks.
Figure 5.6: DDQN Bipedal Walker box plots showing average cumulative reward across 10 episodes for each attack method. Plots (a), (b) and (c) are attacked with a horizon of 5 time steps with budget value of 3, 4, and 5 respectively. (d), (e), and (f) are attacked with a horizon value of 10 time steps with budget value of 3, 4, and 5 respectively.

5.8.5 Comparison of Temporal Projections in LAS

In this section, we present additional visualizations to further understand why $\ell_2$ temporal projections results in more severe attacks in compared to $\ell_1$ temporal projections. Figure 5.7, 5.8, 5.9 and 5.10 presents the $\|\delta_t\|$ usage plot across 100 time steps for both PPO and DDQN in the Lunar Lander and Bipedal-Walker environment. The left subplot represents $\ell_1$ projections in the spatial dimension while the right subplot represents $\ell_2$ projections in the spatial dimensions. These plots directly compare the difference in amount of $\|\delta_t\|$ used between $\ell_1$ and $\ell_2$ temporal projections for both $\ell_1$ and $\ell_2$ spatial attacks.

In most cases with the exception of Figure 5.9, we see a clear trend that $\ell_1$ temporal projections results in a sparser but more concentrated peaks of $\|\delta\|$ utilization (corre-
Figure 5.7: Comparison of $||\delta_t||$ used across time for a single episode in PPO Lunar Lander for different spatial projections with $\ell_1$ and $\ell_2$ temporal projection. Left plot illustrates $\ell_1$ spatial projection and right plot shows $\ell_2$ spatial projection. In both plots, the magnitude of attacks with $\ell_1$ temporal projection attacks dropped to zero from time step 30. However, the magnitude of attacks in $\ell_2$ temporal projection remains high through the episode. Hence, we observe that $\ell_1$ temporal projections essentially allows the agent sufficient time to recover from earlier attacks. In the case of Lunar Lander, the agent might prevent a severe crash while landing or recover from a horizontal thruster boost induced by the attack.

In contrast, $\ell_2$ temporal projections results in a more distributed but frequent form of $||\delta||$ utilization (corresponding to more frequent but weak instances of attacks). We note that while $\ell_1$ projections produces stronger attacks, there is a diminishing return on allocating more attacks to a certain time point as after a certain limit. Hence, this explains the weaker effect of $\ell_1$ temporal projections since it concentrates the attacks to a few points but ultimately gives time for the agent to recover. In contrast, $\ell_2$ temporal projections distributes the attacks more frequently that causes the agent to follow a diverging trajectory that is hard to recover from.

As an anecdotal example in the Lunar Lander environment, we observe that attacks with $\ell_1$ temporal projection tend to turn off the vertical thrusters of the lunar lander. However, due to the sparsity of the attacks, the RL agent could possibly be fire the upward thrusters in time to prevent a free-fall landing. With $\ell_2$ temporal projections, the agent is
Figure 5.8: $\|\delta_t\|$ usage plot of DDQN agent in Lunar Lander across time for a single episode. Left subplot illustrates $\ell_1$ spatial attacks while right subplot shows $\ell_2$ spatial attacks. In each subplot, attacks with $\ell_1$ time projection attacks exhibit periodic spiked patterns while $\ell_2$ time projection attacks are constantly activated with $\|\delta_t\|$ never reaching zero. Since $\ell_1$ time projection attacks periodically inject $\delta_t$ into nominal actions of the DDQN agent, the agent has the opportunity to recover from the attacks.

Attacked continuously. Consequently, the agent has no chance to return to a nominal state and quickly diverges towards a terminal state.

5.8.6 Ablation Study

For this section, we present an ablation study to investigate the effect of different budget and horizon parameters on the effectiveness of LAS vs MAS. As mentioned in the main manuscript, we take the difference of each attack’s reduction in rewards (i.e. attack - nominal) and visualize how much rewards LAS reduces as compared to MAS under different conditions of $B$ and $H$. Figure 5.11 illustrates the ablation study of a PPO agent in Lunar Lander. The figure is categorized by different spatial projections, where $\ell_1$ spatial projections are shown on the left figure while $\ell_2$ spatial projections are shown on the right. Both subplots are shown for $\ell_2$ time projection attacks. Each individual subplot shows three different lines with different $H$, with each line visualizing the change in mean cumulative reward as budget increases along the x-axis. As budget increases, attacks in both $\ell_1$ and $\ell_2$ spatial projection shows a monotonic decrease in cumulative rewards. Attacks
Figure 5.9: $\|\delta_t\|$ usage plot for PPO agent in Bipedal-Walker across time for a single episode. Left subplot illustrates $\ell_1$ spatial attacks while right subplot shows $\ell_2$ spatial attacks. In this figure, both $\ell_1$ and $ell_2$ are seemingly well distributed, although the magnitude of $\|\delta_t\|$ used for both projection schemes are evidently lesser than the other agents in other environments. We speculate that this is due to the nature of the policy learnt by the PPO agent. In this environment, the PPO agent has learnt a policy to operate the Bipedal Walker by bending a knee joint and using the other knee joint to drag itself forward. Hence, in this situation, the agent has learnt a strong stable walking gait and there is not much room for $\delta_t$ to be applied.

In each spatial projection with a $H$ value of 5 shows different trends, where $\ell_2$ decreases linearly with increasing budget while $\ell_1$ became stagnant after $B$ value of 3. This can be attributed to the fact that the attacks are more sparsely distributed in $\ell_1$ attacks, causing most of the perturbations to be distributed into one action dimension. Thus, as budget increases, we see a diminishing return of LAS attacks since attacking a single action dimension beyond a certain limit doesn’t decrease reward any further.

The study was also conducted for PPO Bipedal-Walker and both DDQN Lunar Lander and Bipedal-Walker as shown in Figure 5.13, 5.12 and 5.14 respectively. We only consider attacks in $\ell_2$ temporal projection attacks for both $\ell_1$ and $\ell_2$ spatial projections. At a glance, we see different trends across each figures due to the different environment dynamics. However, in all cases, the decrease in reduction of rewards is always lesser than or equals to zero, which infers that LAS attacks are at least as effective than MAS attacks. We observed that attacks for horizon value of 5 becomes ineffective after a certain budget value. This shows that after some budget value, MAS attacks are as effective as LAS
Figure 5.10: $\|\delta_t\|$ usage plot for DDQN agent in Bipedal-Walker across time for a single episode. Left subplot illustrates $\ell_1$ spatial attacks while the right subplot shows $\ell_2$ spatial attacks. A trend similar to Figure 5.8 is observed where the $\|\delta_t\|$ are utilized in sparse and concentrated instances for $\ell_1$ temporal projections compared to $\ell_2$ temporal projections.

Figure 5.11: Ablation study for PPO Lunar Lander showing effectiveness of attacks comparing LAS with MAS for $\ell_2$ time projection attacks. Left and right subplot shows $\ell_1$ and $\ell_2$ spatial projection respectively. The lines represent different horizon values, where budget is increased along each horizon. Both $\ell_1$ and $\ell_2$ spatial projection scales monotonically with increasing budget.

attacks because LAS might be operating at maximum attack capacity. Interesting to note that Bipedal-Walker for PPO needed a higher budget compared to the DDQN counterpart due to the PPO being more robust to attacks.

5.8.7 Effect of Horizon Parameter in LAS

In this section, we further describe the effect of horizon parameter $H$ on the effectiveness of LAS attacks that we empirically observed. $H$ defines a fixed time horizon (e.g., steps in DRL environments) to expend a given budget $B$. For a fixed $B$ and short $H$, LAS
Figure 5.12: Ablation study for DDQN Lunar Lander showing effectiveness of attacks comparing LAS with MAS. Left subplot shows $\ell_1$ spatial attacks while right subplot shows $\ell_2$ spatial attacks. Both plots are in $\ell_2$ temporal projection attacks. Different lines represent different horizon values, where budget is increased along each horizon.

Figure 5.13: Ablation study for PPO Bipedal-Walker showing effectiveness of attacks comparing LAS with MAS. Left subplot shows $\ell_1$ spatial attacks while right subplot shows $\ell_2$ spatial attacks. Both plots are in $\ell_2$ temporal projection attacks. Different lines represent different horizon values, where budget is increased along each horizon.

favors injecting stronger perturbations in each step. Hence, we would intuitively hypothesize that given a shorter $H$, the severity of LAS attacks will increase as $H$ decrease, as shown by the reduction in rewards between MAS and LAS in Figure 5.3 of the main paper. In most cases, the reduction is negative, hence showing that LAS attacks are indeed more severe. However in some cases as shown in Figure 5.8, 5.9, & Figure 5.10, a shorter $H$ does result in LAS being not as effective as a longer $H$ (though still stronger than MAS as evident from negative values of y-axis). This can be attributed to the nonlinear reward
function of the environments and consequent failure modes of the agent. For example, attacks on Lunar Lander PPO agent causes failure by constantly firing thruster engines to prevent Lunar Lander from landing, hence accumulating negative rewards over a long time. In contrast, attacks on the DQN agent causes Lunar Lander to crash immediately, hence terminating the episode before too much negative reward is accumulated. Thus, while the effect of $H$ on LAS attacks sometimes do not show a consistent trend, we it is a key parameter that can be tuned to control the failure modes of the RL agent.

5.8.8 Action Space Dimension Decomposition

We provide additional results on using the LAS attack scheme as a tool to understand the RL agent’s action space vulnerabilities for a DoubleDQN agent in both Lunar Lander and Bipedal Walker environment. It is interesting to note in Figure 5.15, even with agents trained with a different philosophy (value-based vs policy based, shown in main manuscript), the attack scheme still distributes the attack to a similar dimension (Up-
Figure 5.15: Magnitude of attack with respect to different episodes for Lunar Lander environment with DDQN RL agent. The two colors (Blue and Green) in the bar plots represent the attacks allocated to the two action space dimensions, Up-Down action and Left-Right action, respectively. The attack schema is LAS attack with a budget of 4 and Horizon of 5.

Down action for Lunar Lander), which highlights the importance of the that particular dimension.

In Figure 5.16, we show the outcome of LAS attack scheme on Bipedal Walker environment having four action space dimensions. The four joints of the bipedal walker, namely Left Hip, Left Knee, Right Hip and Right Knee are attacked in this case, and from Figure 5.16, we see that the left hip is attacked more than any other action dimension in most of the episodes. This supports our inference that LAS attacks can bring out the vulnerabilities in the action space dimensions (actuators in case of CPS RL agents).
Figure 5.16: Magnitude of attack with respect to different episodes for Bipedal Walker environment with DDQN RL agent. The four colors (Blue, Green, Red and Cyan) in the bar plots represent the attacks allocated to the four action space dimensions, Left Hip, Left Knee, Right Hip and Right Knee actions, respectively. The attack schema is LAS attack with a budget of 4 and Horizon of 5.

5.9 References


CHAPTER 6. DIRECT 3D PRINTING OF MULTI-LEVEL VOXEL MODELS

A paper to be submitted in Additive Manufacturing
Sambit Ghadai, Anushrut Jignasu and Adarsh Krishnamurthy

6.1 Abstract

We present a direct method for the additive manufacturing of multi-level voxelized models that achieves a better surface finish. We have developed a new multi-level marching squares algorithm to identify the boundary of the multi-level voxelized model. We have also developed methods to use the multi-level voxelization to perform the infill operation based on user-defined infill density. We directly generate the GCode that is input into the 3D printer for printing. Our method overcomes the issues associated with the slicing operation for standard CAD models. In addition, we can directly print thresholded voxel models that are output from CT or MR scans to get a physical 3D representation of medical data. We show that our method performs well by directly printing test models of multi-level voxel representation of complex CAD geometries, and cardiac CT data.

6.2 Introduction

Additive manufacturing (AM, also known as 3D printing) is a process by which virtual CAD models are physically manufactured by adding material instead of traditional
subtractive manufacturing. Most of the AM processes use tessellated CAD models in the form of stereolithography (STL) or virtual reality modeling language (VRML) file format. Tessellated CAD models, in the form of triangle soups, can be easily obtained from generic CAD models, which are usually represented using boundary representation (B-rep) or constructive solid geometry (CSG) modeling of primitives. These tessellated models are further processed or sliced to obtain the layer-by-layer information for AM. However, other data representations used to represent solids, such as volumetric representations (voxels), point clouds, and medical imaging data (MRI and CT Scans), require complex geometric algorithms to tessellate and represent them as triangular facets. Additionally, developing AM printing strategies for these representations is challenging due to the computational cost of processing these representations at higher fidelity. In this paper, we develop an AM strategy that enables direct 3D printing from volumetric representations or multi-level voxel models with higher accuracy and print quality. One of the advantages of using a voxel representation for AM is that the regularity of the voxel grid eliminates the need for explicit slicing which is a time consuming process. Slicing of CAD models also creates gaps and discontinuities that needs further processing to be used for AM. Voxelization alleviates this issue by directly generating a rectilinear occupancy grid conforming to the required layered structure. In addition, it also enables direct printing of models that are natively represented using voxels such as 3D imaging data from MR and CT scan image stacks.

Voxel representations are rectilinear structured grids with scalar values representing the volume elements of a solid geometry. Capturing the fine details of a solid model requires a very high-resolution voxel grid. However, high-resolution uniform data structures for voxels require a large amount of memory and are compute-intensive. Specialized data structures such as octrees [7] and multi-level voxels [26], use a hierarchical approach to represent a dense voxel grid using sparser data. Specifically, multi-level voxels
Figure 6.1: Distinction between a regular voxel representation and multilevel voxel representation. (a) shows a regular voxel grid of a turbine CAD model while (b) shows its multi-level voxel representation. The effective representation resolution of both the grids are similar. (a) has a resolution of $12 \times 16 \times 24$ whereas (b) has a coarse resolution of $8 \times 8 \times 12$ and fine resolution of $4 \times 4 \times 4$, hence achieving effective resolution of $32 \times 32 \times 48$.

provide flexibility to choose the voxels’ size and resolution at each hierarchical level. In addition, fast GPU-accelerated voxelization algorithms for multi-level voxels with very low space and time complexity have been developed [26]. Without loss of generality, we use a multi-level voxelization with two hierarchy levels for representing the volume information. These levels—coarse and fine—have a user-defined selection of voxel grid resolution at each level. Specifically, we identify the voxels corresponding to the outside, inside, and the boundary of the object at the coarse voxel level using different scalar values. Each coarse boundary voxel is subdivided into fine voxels with the same scalar values representing the boundary, inside, and outside. The multi-level voxel representation is shown in Figure 6.1, showing the difference with the regular voxel grid.
Multi-level voxel representation provides us with the necessary boundary and inside information to facilitate 3D printing. We build the surface boundary layers from fine voxels and compute the infill layers only at the coarse voxel level. We have developed a multi-level marching squares algorithm (MLMS) that generates high-resolution isocontours from multi-level voxel representation of CAD models to 3D print the boundary. However, printing the infill of a model with such higher accuracy is generally unnecessary as it does not contribute towards the surface finish. Hence, we use the inside information of fine and coarse voxels of the multi-level voxel representation to generate rectilinear infill structure using a hybrid scan-line approach. This approach conservatively extracts the infill lines closer to the boundary of a print layer while sparsely extracting the infill information from the larger coarse inside voxels at the same time. Thus, we achieve both high fidelity at the boundary and sparse infill structure with user-defined sparsity for direct 3D printing a model directly using multi-level voxels.

In addition to balancing the resolution of the isocontours and infill for 3D printing, multi-level voxel representation also allows us to print the infill with variable layer height without explicit slicing. This eliminates the staircase aliasing artifacts on the boundary of the printed part, leading to a better surface finish. In the layer print direction, we print the isocontours based on each fine voxel layer, but only print the infill once for the coarse voxel layer with a higher extrusion value that can be automatically set on the 3D printer nozzle. Thus, we only print the infill once after printing all the isocontours of the fine voxels in the layer. We choose the resolution of the multi-level voxel representation to conform to the variable layer height parameters. The layer heights for boundary and infill structures govern the fine and coarse voxel sizes, respectively. We then voxelize the CAD model to this resolution using a GPU-accelerated multi-level voxelization algorithm. Based on these layer heights, we also generate the GCode instructions to command
the printer to print the final part with variable heights, thus eliminating the need to slice the model.

In this paper, we have developed a method to directly 3D print a CAD model from a multi-level voxel representation. We have developed a multi-level marching squares algorithm, which can be used to 3D print medical data such as MRI and CT scan data without explicit tessellation. Our main contributions include:

- A direct 3D printing method employing multi-level voxel representations of CAD models to accurately 3D print high resolution models.
- A multi-level marching squares algorithm to generate layer-by-layer contours from a multi-level voxel representation of a CAD model.
• A novel infill generation method that uses a hybrid scan-line algorithm to create variable height infill structures from multi-level voxel representations.

• Application of this 3D printing method to fabricate physical models from high resolution CAD and volumetric medical data such as stacks of MR and CT scan images.

A complete outline of our framework is shown schematically in Figure 6.2. We first voxelize the CAD model according to the user-defined layer heights and resolution, and create a multi-level voxel representation consisting of coarse and fine voxels each with its specific boundary and inside voxels (shown in green in Figure 6.2). We then implement the multi-level marching squares algorithm (Section 6.4) on the boundary voxels of both coarse and fine voxels to generate the boundary isocontours (shown in blue). Using the inside voxels of the coarse and fine levels, we generate the infill structure for the model using a hybrid scan-line approach (Section 6.5, shown in red). Finally, we combine the boundary isocontours and infill structure to create the GCode instructions (Section 6.6) using the user-defined layer heights. We show some physical examples of the 3D printed models in Section 6.7.

6.3 Background and Related Work

Given the recent evolution of 3D Printing technology, there has been an increase in research focused on improving various AM processes [19]. Most generic AM processes use computer-aided design (CAD) representation of a part or model to generate special additive 3D printing directives for manufacturing. Fadel and Kirschman [8] mentions that among a plethora of CAD representation formats that have been used for AM, none have been as universal as stereolithography (STL) format, due to its simplicity. An STL file approximates a CAD model using a tessellated (triangular) surface model and is the de-facto representation for 3D printing of CAD models. The primary input to an AM
process is a GCode file generated from the CAD model after performing the slicing operation on the STL file. However, issues like truncation errors and approximation of curved surfaces by triangular facets in tessellated representations introduce some level of inaccuracy [16]. Additionally, Kumar and Dutta [16] mention problems such as inconsistent normals, topological degeneracy, self-intersections, and geometric degeneracy to be associated with the conversion to STL format.

The slicing operation is a necessary part of the layered additive manufacturing process planning, which has a major impact on the surface finish [14, 18, 28], build time [15], and mechanical properties [20], of the manufactured model. There are various slicing algorithms like uniform slicing [5, 6, 27], its variant adaptive slicing [27], and direct slicing [4]. Uniform slicing generates slices with constant layer thickness and has been widely adopted for different kinds of AM processes. However, uniform slicing of relatively large CAD models is computationally intensive, and the process is slow Choi and Kwok [5, 6], with coarse surface features. Adaptive slicing uses machine capability and geometry Kulkarni and Dutta [14], Ma et al. [18], Zhou et al. [28] to determine slice thickness allowing for reduced build times and a greater surface finish. Zhou et al. [28] proposes the use of non-uniform cusp heights for higher slicing efficiency. In this paper, we perform AM operations using voxel representations of CAD models to overcome the aforementioned issues associated with using tessellated representations and explicit slicing of triangular facets. Regular rectilinear voxel grids have implicit layered information that enables us to perform layered manufacturing directly without undergoing standalone slicing operations. In addition, we perform a variable layer height 3D printing for the surface and inside of the model using our multi-level voxels.

Voxelization is a traditional volume representation schema for CAD models that stores the model’s occupancy information in a regular 3D rectangular grid with volume elements called voxels. Voxelized representations offer the ability to conserve volumetric
Figure 6.3: In-depth view of a multi-level voxel grid. (a) shows a scooby CAD model which is voxelized to get multi-level voxelization as shown in (b). Green and red cubes represent the larger coarse voxels and smaller finer voxels at the boundary respectively. A single coarse voxel layer is extracted in (c) and its top and side views are shown in (d) with detailed fine voxels. (e) shows the fine voxel occupancy values of a single coarse voxel in the x-y and z directions.

data associated with a CAD model [13] and allow for efficient Boolean operations and collision detection of CAD models. The conservation of volumetric data associated with a CAD model is of particular importance for 3D printing [2, 9, 13, 23–25]. However, an accurate voxel representation of a CAD model requires a very large number of voxels (∼ 1 billion) to represent the fine details of the model. This comes with very high computational and memory cost.

To overcome this high memory cost, traditional volume representations have been modified to use various hierarchical data structures such as octrees, and kD-trees. The Octree representation offers efficient memory allocation by using successive subdivisions of an object array into octants [12, 21]. kD-trees, on the other hand, alternatively divide the space along the three principal axes, partitioning the space and allowing for memory-efficient handling of location queries [3]. Laine and Karras [17] uses a sparse octree data structure where each node is represented as a voxel for ray casting. The sparse octree-based approach has also been used by Schwarz and Seidel [22] to create a GPU-based
solid voxel representation while addressing the high memory consumption of traditional voxel grids. Similarly, Young and Krishnamurthy [26] uses the high computation capability of GPUs to generate a multi-level voxelization scheme that stores the occupancy information of a CAD model at two hierarchical levels of voxels. First, a general voxelization is performed on a boundary representation (B-rep) model to store the inside-outside and boundary occupancy information. Then, they further voxelize the boundary voxels into further smaller voxels and maintain the hierarchical relationship of both voxel levels using a prefix sum address data structure.

In this paper, we adopt the multi-level voxelization paradigm developed by Young and Krishnamurthy [26] to easily and rapidly voxelize models with an effective higher resolution, thus capturing the finer details of the CAD model. Further, multi-level voxel representation provides us with two distinct voxel hierarchy levels, which allows us to easily handle the boundary and infill generation for the AM process in a distinct fashion to achieve high surface printing accuracy while sparsely printing infill with a variable height and density.

6.4 Boundary Extraction from a Multi-Level Voxel Model

In this section, we describe our methodology to directly print the boundary of the CAD model from the multi-level voxel representation. We first voxelize the B-rep solid model to create a multi-level voxel representation using the GPU-accelerated multi-level voxelization algorithm developed by Young and Krishnamurthy [26]. The multi-level voxelization can also be created from other 3D model representations such as point clouds and medical imaging data (see Section 6.7). We make use of two independent methods to extract and directly 3D print the boundary and the infill separately. The voxels having boundary information in each of the coarse and fine voxel resolution, $R_c$ and $R_f$ respec-
Figure 6.4: Step-by-step implementation of Multi-level Marching Squares algorithm. (a) shows a coarse layer $L_c$ in x-y direction. The fine boundary voxels are shown with a red fill in (b). (c) shows an example of coarse voxel with its boundary augmented with neighboring fine voxels and the final grid formed is shown in (d). Multi-level isocontour resulting from applying vanilla marching squares on the final grid is shown in (e).

tively, are encoded using a scalar value 1.0 and the empty voxels are encoded using a scalar value 0. Similarly, the voxels representing the inside of the CAD model are encoded with a scalar value 0.5. The multi-level voxelization algorithm also creates a prefix-sum index array that maps each of the course boundary voxels to its specific fine voxels. Please refer to Young and Krishnamurthy [26] for more information about the multi-level voxel representation.

To generate the 3D print directives for the boundary of a voxelized CAD model, we first isolate the layer information inherent in the voxel grid along the print direction. We then implement a marching squares based approach on each layer of the multi-level voxel grid to efficiently extract the isocontour information. We have developed a multi-level marching squares (MLMS) algorithm that extracts isocontours from the boundary of multi-level voxel layers and combines them to generate the boundary printing information. Using MLMS on multi-level voxel grids allows us to create accurate isocontours from voxel grids with very high effective resolution ($\sim$ 8 billion voxels). Using the isocontours, we then generate the GCode for the 3D printer to print the boundary of the CAD model.
6.4.1 Multi-level Marching Squares

The multi-level marching squares algorithm takes in a single layer of multi-level voxels in a particular orthogonal slice of the voxel grid in the print direction. The hierarchical layers in a multi-level voxel model is shown in Figure 6.4. In this model, we consider the Z direction of the voxel grid as the print direction and the X-Y plane as the print layer. We first extract a coarse plane or layer of coarse voxels $L_c$ in the Z direction, from the multi-level voxel grid. This coarse voxel layer further has multiple fine voxel layers defined by the fine voxel resolution $R_f$. We then extract a fine layer of voxel from $L_c$, namely $L_f$, that can be considered as a binary image with varying pixel resolution, as shown in Figure 6.4(b). Since we have information regarding all the boundary voxels in the multi-level voxel grid, we only apply MLMS on the boundary voxels i.e. voxels with a scalar value of 1.0, thus exploiting the sparsity of a multi-level voxel grid. The boundary voxels on a multi-level voxel layer are hierarchically structured with each coarse boundary voxel $B_c$ having further subdivision with fine boundary voxels $B_f$ as shown in Figure 6.3.

Once we extract a fine layer of voxels $L_f$, we perform the standard marching squares (MS) algorithm individually on each of the coarse boundary voxels $B_c$ that is in itself a voxel grid $G_f$ consisting of fine voxels. Implementation of the standard marching squares algorithm involves visiting each of the fine voxels of $G_f$ sequentially along the $x$ and $y$
direction (scanning or marching) with a filter of size $2 \times 2$. The filter compares the current scalar values of $G_c$ with the standard marching squares look-up table of topological cases and draws an isocontour line intersecting the grid edges. The isocontour lines are drawn from the mid-points of the grid edges depending on the topology of the MS filter. The MS algorithm creates a set of such isocontour lines per layer. It is important to note that each of these isocontour lines is directional i.e., we preserve the sequence of the start and endpoints of the isocontour line. This later allows us to easily form a chain of such lines to create an isocontour for a particular profile in a voxel layer as described in Section 6.4.2

One of the main challenges in performing MS operation individually on each $B_c$ is that it creates a disconnect between the neighboring $B_c$ as shown in Figure 6.5. Hence, we pad each of $G_f$ with a single line of fine voxels from its neighboring boundary voxel extremities. This is shown in Figure 6.4(c) where we increase the grid size of $G_f$ by augmenting the grid with its neighboring grid values at the extremes in $\pm x$ and $\pm y$ directions. This creates an overlapping grid between the fine boundary voxels of a voxel layer and ensures that the isocontour generated from standard MS on $G_f$ is continuous along the object boundary. We perform the boundary augmentation based on the coarse voxel index of $B_c$ to determine the neighboring voxels constituting the boundary.

### 6.4.2 Combined Isocontour Creation

The standard MS algorithm on individual coarse boundary voxels $B_c$ in a fine layer $L_f$ creates a set of isocontour lines $iso_f$ for that particular layer. Once the complete $iso_f$ is generated for $L_f$, we check for duplicated isocontour lines in $iso_f$ and remove the entries to have unique isocontour lines in the set. This is done to avoid adding multiple overlapping segments at each grid’s extreme edge, which in turn affects the 3D printing of the boundary. Duplicate isocontour lines are a result of augmenting $G_f$ with its neighboring
grid extremities due to every two adjacent grids sharing a single line of fine voxels, as shown in Figure 6.4(c).

To combine the set $iso_f$ into isocontours, we create a hash table that maps the endpoints of each isocontour line in $iso_f$. Since each mapping defines an isocontour line segment and each line segment has exactly two endpoints, the hash table mapping is bi-directional. Using this mapping, we then generate a chain of connected isocontour line segments, and a set of such isocontours form the boundary of the layer $L_f$. Figure 6.6 shows the isocontours of a fine voxel layer extracted from the boundary voxels of a coarse voxel layer of the Scooby CAD model. We repeat the steps for generating isocontours from each $L_f$ layer and further from each $L_c$ layer, thus generating the isocontours for the whole model (shown in Figure 6.7). Algorithm 7 shows all the steps of the MLMS algorithm.

6.5 Infill Generation for a Multi-Level Voxel Model

In this section, we present an infill 3D printing scheme for multi-level voxels. Several research works have recently focused on optimizing the infill pattern for efficiently print-
Aremu et al. [2] describe a topology optimization framework using voxels to determine the lattice structure for 3D printing of infill. However, since this is not the main focus of our work, we describe a straightforward infill approach using a standard infill pattern. We describe an algorithm that makes efficient use of the multi-level voxel grids for this purpose. We use a rectilinear pattern, computed from the boundary of the multi-level voxel representation of the CAD model, to enable direct infill printing from voxels. However, generating the infill using a multi-level voxelization is not straightforward, since the fine level at the boundary of the models needs to be accounted for during the infill generation process. Using our approach, we can still control the infill density directly, allowing for user-defined model density.

To compute the rectilinear infill pattern, we use a hybrid scan-line approach on both coarse voxel layers $L_c$ and fine voxel layers $L_f$ of the multi-level voxel grid in an adaptive technique. We first scan through $L_c$ in one of the $x$ or $y$ directions, till we encounter
Algorithm 7: Multi-level marching squares (MLMS) algorithm

Input: Multi-level voxel grid, $G_m$
Result: Multi-level Isocontours

1. foreach Coarse Layer $L_c \in G_m$ do
2.     foreach Fine Layer $L_f \in L_c$ do
3.         foreach Coarse boundary voxel $B_c \in L_f$ do
4.             Get fine voxel grid $G_f$ from $B_c$
5.             foreach Neighbor voxel of $B_c$ do
6.                 if Neighbor voxel is boundary then
7.                     Augment $G_f$ extremes with 1
8.                 else
9.                     Augment $G_f$ extremes with 0
10.                end
11.            end
12.       iso_f = VanillaMarchingSquares($G_f$)
13.     end
14.     isocontours = JoinContours(iso_f, hashmap(iso_f))
15. end
16. end

a coarse boundary voxel $B_c$, which in turn has a fine voxel grid $G_f$ associated with it. Adapting to $G_f$, we then scan through its top $L_f$ layer only at the center voxel line as shown. When the scanning encounters a fine boundary voxel $B_f$, we compute its center-point coordinates and append it to a data structure storing the continuous line segments of the scan-line. We then continue scanning along the direction and append points in $B_f$ where the scalar value of the voxels $> 0.5$ (‘inside’ voxels). Once $B_f$ is scanned, the scanning proceeds to the next coarse voxel and checks if it is a boundary or an inside voxel. The above steps are repeated in the case of a boundary voxel. However, if an inside voxel is scanned, the center point of the coarse voxel is added to the scan-line data structure. This continues till another boundary voxel is encountered, in which case the scanned line is extracted and the scan is resumed in the same direction till it reaches the
Figure 6.8: Infill generation using hybrid scan-line in a multi-level voxel grid. An intermediate voxel layer is shown in (a) with coarse and fine voxels divided into inside and boundary voxels. A single scanning of a line of coarse voxels is represented by the green box. (b) shows the top and side views of the hybrid scan-line algorithm extracting infill line adaptively from fine and coarse inside voxels. Side view in (b) shows the layer height with which the extracted infill pattern is printed.

One of the advantages of using this hybrid scan-line approach to determine the infill pattern is the accurate encapsulation of the infill by the boundary. Due to the presence of fine voxels at the boundary of the coarse voxel grid, the scan-line is accurately computed to match the object’s boundary isocontour extracted in Section 6.4. Besides, this algorithm provides a control on the accuracy of the infill in $z$ (layer) direction while enforcing the accuracy in $x$-$y$ (layer plane) direction. In most of the additive manufacturing applications, the infill is required to be sparser than the surface boundary, which we can easily control using our algorithm by selecting the algorithm to scan every fine layer or a single fine layer in the multi-level voxel grid. We control sparse and dense infill pattern by selective scanning for inside voxels. For sparse infill, we scan a subset of the lines instead of all lines in the voxel grid. In addition, we change the direction of scanning for each infill layer by alternating between $x$ and $y$ directions. This is shown in Figure 6.9 with both $x$
Algorithm 8: Hybrid scan-line algorithm on multi-level voxel grid

Input: Multi-level voxel grid, $G_m$

Result: Infill Pattern

1. foreach Coarse Layer $L_c \in G_m$ do
   2. foreach index $y \in L_c$ do
      3. foreach index $x \in L_c$ do
         4. Determine voxel: $v(index_x, index_y)$
         5. if $v = 1.0$ then
         6. Extract fine grid: $G_f$
         7. Fine layer $L_f = \text{TopLayer}(G_f)$
         8. foreach index $x \in L_f$ do
            9. foreach index $y \in L_f$ do
               10. Determine fine voxel: $v_f(index_x, index_y)$
               11. if $v_f \neq 0$ then
                  12. Calculate center point, $P$
                  13. AddPointToLine($P$)
               14. end
            15. end
            16. end
         17. if $v = 0.5$ then
            18. Calculate center point, $P$
            19. AddPointToLine($P$)
         20. end
      21. end
   22. end
23. end

and $y$ scanning shown separately. After combining the layer, we generate a rectilinear infill pattern with the required density. Figure 6.10 shows both sparse and dense rectilinear infill patterns generated from a layer of scooby model.

6.6 GCode Generation

Once we compute the boundary isocontours and infill structures using the MLMS and the hybrid infill algorithm, we generate the GCode directives required to directly 3D
print the multi-level voxel model. We implement a variable layer infill printing to enable efficient layer-by-layer printing by exploiting the embedded layer information from the multi-level voxels. Selecting an appropriate voxel resolution for the coarse and fine level voxels is the key to perform a variable layer infill and boundary printing. We develop the GCode directives for a Fused Deposition Modeling (FDM) 3D printer in this work. We note that we do not discuss the printing of support structures for overhangs in the model in this work. However, the multi-level voxel representation provides all the information required to generate such support structures, which is a possible future direction for this research.

In FDM 3D printing, one of the important properties that define a printed model’s quality and rigidity is the nozzle diameter of the printer’s extruder (hot end). Other properties, such as layer height ($H_l$), extrusion width ($W_e$), feed rate, traversal speed, and material consumption, are defined by the nozzle diameter [1, 10, 11]. In general, a printed model’s layer height is selected to be within a range of the nozzle diameter for better print quality. Further, the infill pattern is designed to consume less material as it does not contribute to surface quality. We utilize these properties to selectively define the coarse voxel resolution as $N_{x1} \times N_{y1} \times H_l$ where $N_{x1}$ is number of voxels in x direction, $N_{x2}$
Figure 6.10: Low and high density infill pattern visualizations in scooby model. (a) and (b) show the 2D view of the infill visualized from the GCode. (c) and (d) show sectional views of 3D printing simulation of the GCode. (a) and (c) are infill structures with low infill. (b) and (d) are infill structures with high density infill.

is number of voxels in y direction and layer height $H_l$ is the number of voxels in z (print direction). We also define the fine voxel resolution as $N_{x2} \times N_{y2} \times 3$ where $N_{x2}$ and $N_{y2}$ are the number of fine voxel in x-y directions in a coarse boundary voxel. Hence, the effective resolution of our multi-resolution voxel grid is $(N_{x1}N_{x2}) \times (N_{y1}N_{y2}) \times (H_l/3)$. In addition to providing a higher resolution in the x-y direction, we achieve a single layer height of $H_l/3$ due to the subdivision of the coarse voxels into three fine voxels in z direction.
We define the print directives for boundary isocontours generated in Section 6.4 with this layer height of $H_l/3$ while the infill pattern generated in Section 6.5 have a height of $H_l$. We select $H_l/3$ as the layer height (as shown in Figure 6.11) since it enables us to capture finer details of the model. It allows a relatively small layer height to print the boundary with precision while having a large enough layer height to print the sparse infill structures without compromising the structural integrity. We print the infill layer of a higher thickness at each third layer of the boundary layer with this printing protocol.

To generate the GCode for FDM 3D printing, we compute the extrusion values of the print material based on the layer heights, $H_l/3$ and $H_l$, and the distance between two consecutive points in the isocontours and infill patterns respectively. Printer specific instructions in the GCode, such as motor control, extruder temperature, external cooling fan control and bed temperatures, are generic based on the printer and material used. We first compute the $X, Y$ coordinates of point in the isocontour to be printed and define the $Z$ coordinated based on the layer index value of the respective isocontour. Further, we compute the extrusion value, $E$ and the feed rate $F$ and define it in the GCode. Then we follow the process for all the isocontour points and profiles in the layer. This is repeated for all the $L_f$ in $L_c$. After that, we proceed to follow the print process for the infill pattern of $L_c$ with the appropriate $E$ value computed from $H_l$. In addition, we swap the scan
Figure 6.12: 3D Print simulation for Scooby model. (a) shows the final 3D print simulation of the model with $64 \times 64 \times z$ coarse and $8 \times 8 \times 3$ fine voxel resolutions. (b) shows cross-sectional view of the simulation with linear infill pattern.

direction of hybrid scan-line algorithm every consecutive layer to have alternating linear infill patterns for each layer. We follow this routine for all the $L_c$ and $L_f$ layers in the voxelized model to achieve the final GCode ready for printing.

6.7 Results

To demonstrate the direct 3D printing capability from multi-level voxels, we implement the complete framework on a variety of CAD models, including tessellated models and medical imaging data (CT scans). We perform the GPU-accelerated multi-level voxelization of CAD models to get three separate data structures representing the coarse voxels, fine voxels, and a prefix sum address array that stores the fine voxel index values for each coarse boundary voxel. We use these three sets of information to perform di-
rect 3D printing operations and get the GCode directives to finally print a model using a MakerGear M2 desktop 3D printer. The MLMS and the hybrid scan-line algorithms are implemented in a Python environment; the GCode generation framework will be published as open-source online.

To verify the printability of our framework, we simulated the 3D printing process using the generated GCode in open-sourced Ultimaker Cura software. Figure 6.12 shows the final 3D printed simulation of a Scooby model which is voxelized to a coarse resolution of $64 \times 64 \times z$ and fine resolution of $8 \times 8 \times 3$ thus producing an effective resolution of $512 \times 512 \times 1000$ voxels. Similarly, Figure 6.13 and Figure 6.14 shows the direct 3D printing simulations of the stanford bunny and a turbine blade model respectively. Due to such a high resolution representation of voxels, we can observe that in the $x-y$ direction (print layer plane) the isocontours generated from fine voxels overcome the staircase effect that is generally associated with voxel grid isocontours. Specifically in the $z$ direc-
tion (print direction), the staircase effect is non-existant and a smooth surface finish is obtained.

In Figure 6.12(b), the alternating pattern in the infill layer can be observed where each infill layer prints either along the horizontal or the vertical direction. Figure 6.13(b) further shows a cross-sectional view of the print simulation where there are two separate topological profiles of the model with a singular infill pattern closely conforming to the respective boundaries. The final 3D printed models are shown in Figure 6.16. Final 3D printed models using multi-level voxels.

Figure 6.15 shows a direct 3D printing rendering obtained from a stack of CT-scan images of the human heart. The CT-scan images for the heart model are stacked on top of one another, and we create a voxel grid. The voxel grid has inside-outside occupancy information of the heart model as shown in Figure 6.15(a) with a resolution of 512 × 512 × 116. We create a multi-level voxel grid by performing a convolution operation on the original grid to create a coarse voxel grid (resolution of 64 × 64 × 116) and a fine voxel grid (resolution of 8 × 8 × 4). We copy each fine layer of voxels based on each image slice’s
thickness provided in the CT-scan data. This preserves the size and aspect ratio of the model, and the final multi-level voxel grid has an effective resolution of $512 \times 512 \times 464$, with 464 fine layers to be printed. Then we use our direct 3D printing framework to generate the GCode information.

### 6.8 Limitations & Future Work

We describe a few limitations of the current work in this section. We did not explore support structure generation for overhang sections in a CAD model. This can be seen in the final 3D printed objects where due to the absence of support structures, we get warped boundaries at overhang locations of the model. However, due to high resolution voxelization using multi-level voxel representation, the generated contours are highly accurate which allows us to 3D print the complete models without incurring heavy penalty on the surface quality of the overhangs.

In addition, the multi-level voxel representation is not readily available for medical data. We pre-processed the high resolution medical data to generate a multi-level voxel representation that is true to the original data in the $x$-$y$ layer plane. However, due to
Figure 6.16: Final 3D printed models using multi-level voxels. (a), (b) and (c) show the 3D printed model of scooby, bunny and turbine models respectively.
stacking up a copy of each image slice in the print direction, there is an obvious staircase effect.

Future work for the presented direct 3D printing framework involves adding additional 3D printing features such as generating support structures for overhangs. Along with the boundary and inside-outside information for the voxel grid, we also store the average triangle normal information for each voxel based on the base tessellated model. This normal information can be used to find overhang regions in the model and generate appropriate support structures. In addition, different infill structure patterns can be explored to be 3D printed using voxel grids.

6.9 Conclusions

In this paper, we have developed an additive manufacturing framework to directly 3D print CAD models from multi-level voxel representation. The multi-level voxel representation has inherent layer information, obviating the need for the slicing operation. We extract accurate isocontours from high-resolution multi-level voxels using our multi-level marching squares algorithm to print the model’s boundary surface. This method exploits the multi-level voxel representation’s sparse nature to heavily reduce the aliasing or staircase effect in the model, achieving a better surface finish. We generate and print the infill structure of the voxel model with a height proportional to the coarse voxels, while the boundary isocontours are printed with layer height proportional to the fine voxels, which further reduces the aliasing effects in the z-direction. In addition, we can also directly control the infill density, leading to prints with user-defined weight. We believe this approach will be widely adopted by the additive manufacturing community due to its flexibility and ease of adapting it to different layered manufacturing processes.
6.10 References


CHAPTER 7. SUMMARY AND DISCUSSIONS

7.1 Summary

In this thesis, we have developed a Cybermanufacturing framework that facilitates an automated, less time-consuming, high quality, and end-to-end manufacturing process with a focus on data-driven predictive decision-making. The CM framework has two primary segments, namely a cyber space and a physical space. We have developed components in each segment to alleviate the issues seen in the traditional manufacturing process and make a significant contribution to Industry 4.0 realization. We developed algorithms to actively convert the data flowing between the segments into meaningful information using deep learning. In addition, we also maintain data flow consistency through the use of voxels for CAD model representation throughout the framework, thus improving interoperability among the components and sub-components of the framework. The specific contributions are mentioned as follows:

1. We first developed a design for manufacturing (DFM) framework incorporating deep learning techniques (DLDFM) to predict the manufacturability of holes in 3D CAD models in Chapter 2. We pre-process or voxelize the CAD model to a voxel representation with binary occupancy data to preserve the model’s volume information. Using this process with GPU-acceleration, we build a dataset consisting of many such models with priors and construct a 3D convolutional neural network to learn the depth-diameter manufacturability rules. The 3D CNN predicts the man-
ufacturability of models with an accuracy of 80%. This will allow designers to efficiently design a part in real-time through a feedback system using DLDFM that predicts the part’s manufacturability.

2. In Chapter 3, we developed a 3D object recognition framework using a multi-level 3D convolutional neural network on multi-level voxel representation of 3D CAD models to enable process planning. We used multi-level voxel representation that captures features of the CAD model at multiple scales efficiently. We developed the multi-level 3D CNN to learn features from this high-resolution hierarchical voxel representation and to predict the class of the CAD models based on these features. We achieved a prediction accuracy of 92% on the ModelNe dataset with significant GPU memory usage reduction. Predicting features at different scales will allow manufacturing engineers to process plan and determine the tools, equipment, and processes to be employed for the manufacturing of the part or product.

3. Chapter 4 of this thesis describes the product visualization and geometry component of the CM framework. In this chapter, we developed a GPU-accelerated collision detection module for navigation of vehicles in enclosed point cloud spaces. We perform collision detection of point clouds of an environment with a voxel representation of a vehicle to determine the vehicle’s navigation ability. Further, we provide clearance around the vehicle with theoretical guarantees regarding the collision with the environment. We also developed a GPU-accelerated voxel-based Minkowski sum to provide additional flexibility for clearance. The complete framework was developed in a game engine that allows a user to directly interact with the enclosed space environment, check for the navigation and collision, and in turn, make effective design changes to a vehicle or the environment.
4. In the CM framework, the physical space deals with actual physical manufacturing and automation of machines. In Chapter 5, we designed optimization-based attack strategies on the action space of reinforcement learning agents in the cyberspace that can actively exhibit the vulnerabilities of such an agent in making decisions. We developed two attack strategies, namely, Myopic action space (MAS) attacks and Look-ahead action space (LAS) attacks, for these purposes. While MAS greedily attacks the agent at each time step, LAS looks into the future to decide the attacks along a trajectory. We implemented these strategies on multiple RL environments and analyzed the results for attacks. We found the LAS attacks are more powerful than the MAS attacks as they consider the system’s dynamics while crafting an attack. Further, we took a deeper look into the action space after the attack and found that vulnerabilities of particular joints in an actuator or robot can be isolated using these.

5. The final sub-component in the CM framework’s physical space is presented in Chapter 6, where we perform the actual manufacturing of a part. We developed a direct 3D printing method to print from multi-level voxel representations of CAD models. In addition, other volumetric data such as medical MRI and CT data can also be directly 3D printed using this method. We developed a multi-level marching squares method to extract iso-contours from a multi-level voxel model. We formulated a hybrid scanline algorithm to generate the less dense infill lines from the voxel model. Finally, we generate the 3D printing instructions or GCode, simulate the printing, and print the actual parts using this method.
7.2 Future Work

This thesis reports the extensive development of a Cybermanufacturing framework. This CM framework can be extended and further developed to make the manufacturing process more flexible and effective. The possible future directions are as follows:

1. The intelligent DFM work described in Chapter 2 can be extended to include other rule-based features present at multiple scales and resolutions. The feature recognition task presented in Chapter 3 and DLDLM can be combined together for this purpose to make the decision-making process more flexible and robust. This can be applied to check the manufacturability of many mechanical parts and components to aid in the design and manufacturing process.

2. In Chapter 3, we developed a feature recognition system to detect multi-scale features in a CAD model to guide process planning in manufacturing. However, this approach is limited by the design of the CAD model and its features. It does not take into account the exact process of manufacturing these design features. A future direction would be to further extend the process planning by introducing the available machines and tools into the framework. Based on the tool-set information, a data-driven approach can be developed to predict the processes’ sequence to be followed for the final manufacturing of a part.

3. We developed a collision detection module in Chapter 4 for efficient navigation of vehicles in enclosed point cloud spaces. The complete framework is developed in a game engine to provide maximum flexibility and control to the user. One future work is to have Virtual Reality (VR) and Augmented Reality (AR) support to visualize the complete environment in a 1:1 scale. VR’s capability to visualize point clouds and voxels in the framework has already been tested during the current re-
search work, which needs to be implemented and enhanced for better user interaction. Another future direction would be to optimize the end-to-end process for real-time collision detection of point cloud-voxels and its visualization.

4. Chapter 5 describes attack strategies on reinforcement learning agents’ action space to determine vulnerable joints and actuators in a robotic environment. This is an optimization-based white-box solution that needs complete information regarding the virtual cyberspace agent to craft the attacks. However, in a real manufacturing environment, white-box attacks are infeasible most of the time. Possible future work for this problem is to explore surrogate models to craft action space attacks with the same or higher severity. The concept of stealth attacks or imperceptible attacks can also be explored more with the budget parameter’s help. In addition, efficient robustification strategies for action space attacks on agents is also a future direction of the research.

5. In Chapter 6, we developed a direct 3D printing strategy to print using multi-level voxel representations of CAD models. We developed algorithms to generate the boundary contour information and the inside infill information to 3D print the voxel model. However, we did not explore the support structure generation of the print model from voxels. Hence, an exciting future direction of this research would be to use the multi-level voxel grids’ surface normal information to generate support structure. This will result in better quality prints from voxels without overhangs or warping. Further, the multi-level marching squares algorithm can be optimized with GPU support to make the process faster.

Apart from these specific future directions of the work presented in the thesis, new components can be added to the CM framework at both the cyber segment and the physical segment. One of the new components can be a data-driven generative model genera-
tion to guide the designers while designing a part. This can be used as a design support tool that will reduce the designer’s time to 3D model a potential design. Another possible component in the physical segment would be to perform additive metal manufacturing using efficient predictive analysis from cyberspace regarding the process, material, design, and analysis factors.

### 7.3 Concluding Remarks

The research work presented in this thesis is conducted with the goal of advancing the current manufacturing processes to adapt to the Industry 4.0 concept with Cybermanufacturing easily. The two basic components, i.e., cyberspace and physical space, and the respective sub-components, attempt to bridge the gap between the soft design phase and physical manufacturing. In addition, the interaction between the two segments with consistent data flow, data-to-information conversion, and the concept of the digital twin in manufacturing attempts to present the complete CM work as a cohesive unit. We hope that this thesis will benefit several other researchers to make significant contributions to Cybermanufacturing.