Computer Vision in Ice Hockey: Realistic Stick Augmentation and Virtual Game Reconstruction

by

Vasyl Chomko

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Applied Science
in
Systems Design Engineering

Waterloo, Ontario, Canada, 2025

© Vasyl Chomko 2025

Author's Declaration

This thesis consists of material all of which I authored or co-authored: see Statement of Contributions included in the thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Statement of Contributions

This thesis incorporates material from one paper for which I was the lead author. I was responsible for the conceptualization, code development, manuscript preparation, and submission.

Chapter 3 contains material from the following previously published work:

Chomko, V., Chen, Y., Clausi, D., & Wong, A. (2024). *Synthetic Local Data Augmentation. 2024 IEEE 26th International Workshop on Multimedia Signal Processing (MMSP).* doi:10.1109/MMSP61759.2024.10743660.

Abstract

Ice hockey presents a uniquely challenging environment for computer vision due to its fast pace, heavy occlusions, motion blur, and the visual similarity of objects like sticks and players. Compounding these issues is a lack of annotated data, which limits the effectiveness of modern deep learning models. This thesis addresses two key problems in this domain: robust detection of hockey sticks under visual noise, and the generation of realistic, annotated video data to support model development and analytics.

First, we introduce **Synthetic Local Data Augmentation (SLDA)**, an instance-level augmentation strategy tailored for hockey stick segmentation. SLDA injects real segmented stick masks into broadcast images using context-aware transformations such as motion blur, geometric scaling, and lighting adjustments. Applied to a custom dataset of over 4,000 stick annotations, SLDA significantly improves segmentation accuracy on occluded and fast-moving sticks, yielding gains of up to +5.8% mAP@50 and +4.0% F1-score over strong baselines.

Second, we present a **Unity-based 3D Hockey Simulation Tool** that reconstructs entire hockey game sequences from tabular puck and player coordinates. The simulator animates realistic gameplay with fully configurable cameras, enabling multi-angle video synthesis and precise ground-truth annotations. This tool is designed to support model training, evaluation, and visual analytics by converting coordinate logs into synchronized multi-view videos paired with dense per-frame annotations—player bounding boxes, player rink coordinates, segmentation masks, and full camera parameters (intrinsics and extrinsics). A full quantitative assessment of these uses is left for future work.

Together, these contributions demonstrate how targeted augmentation and synthetic simulation can overcome real-world data limitations in sports vision. By combining SLDA's image-level enhancements with full-scene reconstruction, this thesis lays a foundation for more robust, scalable, and intelligent systems in automated hockey analytics and beyond.

Acknowledgements

I am deeply grateful to my supervisor, *Professor David A. Clausi*. He gave me room to follow my curiosity and try different ideas, while checking in, answering questions, and keeping me moving when I got stuck. He helped me narrow my interests into a clear, meaningful project in computer vision that truly fit what I care about, and his continuous help turned uncertainty into progress. In the Vision and Image Processing (VIP) Group, he made sure I had the tools and support to do the work, and set an example of rigor, kindness, and leadership that I hope to carry forward. I also thank *Professor Yuhao* for helping me find answers when I didn't know where to start and for pointing me in the right direction when I was unsure. I am additionally thankful to *Professor Alex Wong* for his encouragement and thoughtful feedback.

I wish to express heartfelt thanks to the *Waterloo Artificial Intelligence Institute*, then led by *Harold Goldwyn*, and to *Jeff Casello* for creating and supporting the program that enabled Ukrainians like me to continue our studies during the first months of the war. When my home country was under attack and my future felt uncertain, their support allowed me to come to Canada, settle, and keep learning. While completing my undergraduate studies online at a Ukrainian university, I was able to begin coursework at the University of Waterloo; that experience helped me understand what I truly wanted - to pursue and complete a master's degree!

Thank you to all my colleagues in the *Vision and Image Processing (VIP) Lab*, for our game nights, for being so supportive and open, and for inviting me into your projects and letting me become part of them.

My family has been everything to me. *My father* has always offered the right advice at the right time and showed me the kind of person I want to become. *My mother* and *my sister* have stood by me no matter the distance; even from far away, I always felt their strength behind me. *My uncle* sparked my love for technology and computers when I was a child and taught me to look deeper, not only into systems and code, but also into people. I am endlessly grateful. I also thank my friends in Canada who helped me stay connected to Ukraine and remain resilient. A special thanks to *Kateryna* and *Artem* for the countless late nights preparing for exams and polishing projects; this journey would not have been as meaningful or as joyful without you.

Finally, I thank *Ukraine*, its *Armed Forces*, volunteers, and everyone working for our victory and freedom. Your courage and sacrifice make it possible for students like me to keep learning, building, and hoping. I believe in you, and I dedicate this work to that belief.

Dedication

To my family, whose love made every step possible; to Ukraine and all who defend it; and to the mentors and friends at Waterloo who opened a path for me when I needed it most.

Table of Contents

Author's Declaration ii

Statement of Contributions iii

Abstract iv

Acknowledgements v

Dedication vi

List of Figures x

List of Tables xii

List of Abbreviations xiii

1 Introduction 1

1.1 Motivation and Research Challenges 1

1.2 Problem Statement and Objectives 3

1.3 Overview of Contributions 4

1.4 Thesis Organization 6

2 Background and Literature Review **8**
2.1 Sports Analytics and Hockey Video Analysis Challenges 8
2.2 Data Scarcity in Hockey and the Role of Synthetic Data 10
2.3 Object-Level Data Augmentation Techniques 13
2.4 Context-Aware Augmentation for Occlusion and Motion Blur 16
2.5 Player Tracking, Pose Estimation, and Puck Detection in Hockey 18
2.6 Simulation Tools and Synthetic Data Generation for Sports Analytics . . . 22
2.7 Summary of Literature 26

3 Synthetic Local Data Augmentation (SLDA) **29**
3.1 Introduction 29
3.2 Methods 32
3.2.1 Hockey Stick Catalog Dataset 33
3.2.2 Object-level Augmentation Module 33
3.2.3 Size and Resolution Adjustment Module 34
3.2.4 Image Assembling Module 35
3.3 Experiments 36
3.3.1 Experiment Setup 36
3.3.2 Quantitative Results 38
3.3.3 Qualitative Results 42
3.3.4 Visual Comparison of Different Augmentation Scenarios 43
3.4 Conclusion 45

4 Scene-Generating Simulation Tool (Unity Project) **47**
4.1 Tool Purpose and Functional Overview 48
4.2 System Architecture 49
4.2.1 Input Handling Layer 50
4.2.2 Data Processing Layer 52

4.2.3 Rendering and Output Layer 55
4.3 Detailed Tool Descriptions 56
4.3.1 Manual Homography Estimation Tool 56
4.3.2 Manual Puck Annotation Tool 59
4.3.3 HockeySim Auto-Homography Tool 61
4.3.4 Player Tracking and Annotation Tool 63
4.4 Unity Rendering Engine 64
4.4.1 Launch and Input Specification 65
4.4.2 Rendering Environment and Assets 66
4.4.3 Animation Logic 67
4.4.4 Synchronized Annotation Output 69
4.4.5 Camera Control and Output Options 71
4.5 Usage Workflow 72
4.5.1 Scenario 1: Scene-Generating Data Augmentation from Tabular Input 72
4.5.2 Scenario 2: Recreating Broadcast Footage from Real Game Video . 73
4.6 Rendering Examples 75

5 Comparative Analysis of SLDA and the Unity Simulation Tool 77
5.1 Addressing Data Scarcity and Complexity in Hockey Analytics 77
5.2 Data Augmentation vs. Synthetic Simulation for Model Training 79
5.3 Interoperability and Complementary Use of Both Methods 81
5.4 Strengths, Limitations, and Use-Case Tradeoffs 83

6 Conclusions and Future Work 88
6.1 Summary of Contributions 88
6.2 Future Work 90

References 92

Glossary 99

List of Figures

2.1 Taxonomy of image data augmentation methods. 13

2.2 Occlusion-aware player tracking in ice hockey.(Prakash *et al.*, 2024) 18

2.3 Sports simulation in the Google Research Football environment.(Kurach *et al.*, 2020) 22

3.1 Example of hockey stick segmentation challenges. 30

3.2 Workflow of the SLDA technique for hockey stick augmentation. 32

3.3 Validation Segmentation Loss over Training Epochs, Comparing Different Data Augmentation Scenarios. 41

3.4 Visual comparison of hockey stick segmentation across models. 44

4.1 Frame from a broadcast video used as input for scene reconstruction. . . . 47

4.2 Corresponding frame from the Unity-based simulation output. 47

4.3 Overview of the system architecture. 49

4.4 Rink template used as the coordinate system for all player and puck annotations. 50

4.5 Manual Homography Estimation Tool: Matching points between uploaded image and template 56

4.6 Homography Matrix Generation: Preview of alignment result using visual overlay 58

4.7 Manual Puck Annotation Tool: Interface 59

4.8 HockeySim Auto-Homography Tool: Example of debug overlay 61

4.9 Unity Rendering Engine: scene preview and camera configuration prior to rendering. 65
4.10 Unity Rendering Engine: Hockey rink asset 67
4.11 Unity Rendering Engine: 3D animated player asset with motion clips . . . 67
4.12 Output: Example of segmentation mask of a rendered frame. 70
4.13 Simulation settings for a fixed top-down camera. 76
4.14 Simulation settings for a fixed side-view camera. 76
4.15 Simulation rendering result for a dynamic camera. 76
4.16 Simulation rendering result for a broadcast-like camera. 76

List of Tables

2.1 Summary of augmentation and simulation papers. 28

3.1 Transformation parameters and probabilities. 34
3.2 Traditional Data Augmentation Parameters 37
3.3 Comparison of mAP50 for Different Augmentation Conditions 38
3.4 Comparison of mAP@50–95 for Different Augmentation Conditions. 39
3.5 Performance metrics for YOLOv8s under different data augmentation scenarios. 39
3.6 Comparison of mAP50 Across Architecture Sizes. 42
3.7 Precision, Recall, and F1-Score Across Architecture Sizes. 43

4.1 Input: Example of Game-State Table 51
4.2 Input: Example of Player Metadata Table 51
4.3 Input: Example of Puck Annotation Output 61
4.4 Auto-Homography Tool: Command-line arguments 62
4.5 Player Tracking and Annotation Tool: Command-line arguments 63
4.6 Output: CSV Format for Player Annotations 64
4.7 Unity Rendering Engine: Command-line arguments 66
4.8 Output: Example bounding box table (pixel-space) 70
4.9 Output: Example frame metadata including full camera parameters 71
4.10 Output: Example player positions in rink coordinate system 71

List of Abbreviations

AP Average Precision 14, 38

COCO Common Objects in Context 9, 14, 15, 19, 28

DeepSORT Deep Simple Online and Realtime Tracking 19

DLT Direct Linear Transform 53

FoV Field of View 24

GAN Generative Adversarial Network 13, 15

GNN Graph Neural Networks (GNNs) are deep learning models designed to operate on graph-structured data. 91

GT Ground Truth 10

HockeyOrient Annotated Dataset for Hockey Player Orientation Classification 20

ImageNet Large-scale Visual Recognition Challenge dataset 9

IoU Intersection over Union xiii

mAP@50 mean Average Precision at 50% IoU 3, 5

Mask R-CNN A neural network model that performs object detection and instance segmentation 15, 28

MOTA Multiple Object Tracking Accuracy 11, 28

NHL National Hockey League 7, 10, 19, 24, 28, 36

OCR Automated transcription of text from images or video frames. In sports analytics, OCR is commonly used on jersey numbers to associate tracked players with identities. 20, 91

OpenPose An open-source framework for real-time multi-person 2D pose estimation. 19

SLAM Simultaneous Localization and Mapping (SLAM) refers to the computational problem of constructing or updating a map of an unknown environment while simultaneously keeping track of an agent's location within it. 91

SLDA Synthetic Local Data Augmentation iv, x, 4, 5, 7, 12, 17, 26, 29, 31–33, 36, 38–47, 75, 77–91

SORT Simple Online and Realtime Tracking 19, 28

SOTA state-of-the-art 6, 9, 14, 19, 21, 23, 28

Unity HDRP Unity's High Definition Render Pipeline (HDRP) is a rendering pipeline optimized for high-fidelity graphics, realistic lighting, and advanced visual effects. 91

VIP Lab Vision and Image Processing Lab 25

YOLOv8-seg An extension of the YOLOv8 object detection framework that supports instance segmentation. 31, 38, 45, 88

Chapter 1

Introduction

1.1 Motivation and Research Challenges

Sports analytics has rapidly evolved with the advent of computer vision techniques that can automatically interpret game footage. In professional sports such as ice hockey, automated vision-based analysis offers enormous potential to enhance coaching decisions, player performance evaluation, and the broadcast viewer experience [40]. For example, robust player tracking and object detection can enable detailed statistics (e.g., player speed, possession time) and novel visual augmentations during live broadcasts [40]. However, ice hockey presents a particularly challenging environment for computer vision. It is a fast-paced, dynamic sport played in a confined rink with frequent occlusions and rapid object motions. Broadcast hockey videos often feature motion blur from swift player and puck movement, sudden camera pan/tilt/zoom adjustments, and frequent occlusion of players and equipment as they collide along the boards or in front of the net [40, 46]. These factors make tasks such as object detection and multi-object tracking far more difficult than in many other domains.

A central challenge in hockey video analysis is the small size and high speed of critical objects. The hockey puck is only 7.5 cm in diameter and moves extremely fast (often obscured by motion blur), while hockey sticks are thin elongated objects often overlapping with players. As a result, the puck and sticks can virtually disappear in video frames due to low resolution and occlusions [46]. Prior studies note that puck detection in broadcast hockey footage is "significantly challenging" because of the puck's small size, frequent occlusions by players, and motion blur [46]. Similarly, hockey sticks can be difficult for generic object detectors to recognize, as they may move quickly and appear blurred, be

occluded by the player's body or the boards, or even extend partially outside the field of view. Human annotators and automated models alike struggle with these occluded and fast-moving targets.

Compounding these visual challenges is the **limited availability of annotated hockey image data** for training modern deep learning models. High-performing computer vision models demand large labeled datasets, but in the domain of hockey (and sports in general), such datasets have been scarce or proprietary. Unlike sports such as soccer or basketball which benefit from some public datasets and well-studied benchmarks, hockey has seen relatively few small-scale open datasets until recently. In fact, researchers in this field have had to create new datasets to push progress, for example, Prakash et al. introduced two hockey tracking datasets (HTD-1 and HTD-2) in 2025 because existing data was insufficient [40]. This indicates that acquiring annotated hockey data is labor-intensive and not many extensive datasets are readily available to the community. The lack of data is a fundamental barrier, since models trained only on general images or other sports do not perform well on hockey's unique visual conditions.

The combination of data scarcity and intrinsic visual difficulty motivates the exploration of approaches that can **amplify training data and inject domain-specific knowledge** into the model training process. Data augmentation and synthetic data generation have emerged as powerful strategies to address limited data in many computer vision tasks [6, 21]. By artificially expanding the effective dataset and simulating difficult scenarios (e.g., occlusions, motion blur) during training, one can teach models to handle the very cases that would normally confound them. In the context of hockey, this means finding ways to generate additional **realistic training examples of hockey sticks** in varied poses and conditions, without the prohibitive effort of manual annotation for thousands of images.

Moreover, beyond still-image augmentation, there is a broader motivation to create **interactive simulation environments** for hockey. If we can reconstruct and simulate hockey gameplay in a virtual 3D world, we unlock the ability to generate unlimited annotated visual data from different camera angles, to reproduce specific game situations for analysis, and to facilitate new research (for example, testing tracking algorithms on a known ground-truth scenario, or generating synthetic video for training action recognition models). Simulation can act as a "digital twin" of real hockey games, where the positions of players and objects are known and controllable. Such a system can support training workflows that require multi-view or long-sequence footage that is difficult to obtain from a single broadcast. A formal assessment of usability and downstream impact is left for future work. It also serves educational and analytical purposes, coaches and analysts could replay sequences from any perspective, and researchers could pose "what-if" scenarios in a controlled environment.

In summary, the key research challenges motivating this thesis are: **(1)** the need for improved detection of occluded and fast-moving hockey objects (specifically sticks) despite limited labeled data, and **(2)** the need for tools to bridge the gap between abstract hockey game data (e.g., player trajectories) and the rich visual information of actual gameplay through simulation. Addressing these challenges has the potential to significantly advance automated hockey analytics. By augmenting real data with realistic synthetic examples, we aim to boost model robustness for object detection in hockey imagery. By building a 3D simulation of hockey games, we aim to provide a platform for generating new data and visualizing game dynamics in ways not possible with a single camera feed.

1.2 Problem Statement and Objectives

Given the above challenges, this thesis tackles two interrelated problems in the domain of hockey analytics and computer vision:

Problem 1—Robust Hockey Stick Detection under Limited Data: How can we improve the computer vision detection and segmentation of hockey sticks in broadcast game images, especially when the sticks are partially occluded, motion-blurred, or only small portions are visible? Standard object detection models struggle with these conditions due to scarce training examples capturing such cases. The objective under this problem is to develop a method of *instance-level data augmentation* – specifically tailored to hockey sticks – that can synthetically generate new training examples of sticks in realistic contexts. By increasing the diversity and quantity of stick instances (including difficult cases of occlusion and motion), the goal is to train a detection model that achieves higher accuracy and recall for hockey sticks than training on the available real data alone. In particular, we seek to enhance performance on metrics such as precision, recall, F1-score, and mAP@50, indicating better ability to find sticks in challenging frames.

Problem 2—Reconstructing and Simulating Hockey Game Sequences: How can we create a three-dimensional simulation of a hockey game from available game data, either time-stamped player/puck coordinates or *trajectories extracted from video* to be used as a source of synthetic video data for analysis and model training? Hockey games are increasingly tracked via sensors or manual annotation, yielding rich tabular datasets of player and puck positions over time. However, there is a gap between this **spatiotemporal data** and visual understanding, currently, researchers often analyze such data in abstract forms or 2D plots. The objective in this problem is to build a software tool that takes in structured game data (e.g., (X, Y) positions of each player and the puck over time) and uses

a game engine to reconstruct the play in a virtual 3D rink. The simulation should support *camera manipulation* (e.g., moving to any viewpoint, zooming) and produce rendered video output. In doing so, the tool can generate synthetic video feeds from any angle (including views not captured by the original broadcast) and produce annotated video (since the exact positions of players/puck are known in the simulation). The ultimate objective is to provide a framework that can be used for tasks such as multi-view action recognition (by rendering multiple angles), generating data to train tracking algorithms (with perfect ground truth from the simulation), or visualizing sequences for coaching and analysis. While developing a full research pipeline (from raw video to automatic simulation) is beyond scope, this tool is meant as a **foundational step** toward that goal, enabling future integration of broadcast video analysis (detection, homography, tracking outputs from works by others) with a visualization engine.

These two problems are distinct yet complementary. Both aim to overcome data limitations: the first through *augmenting existing images* (a 2D approach), and the second through *creating new images via simulation* (a 3D approach). Together, they contribute to a larger vision of using synthetic data to advance sports analytics. The specific objectives can be summarized as follows:

Objective 1: Design a context-aware data augmentation pipeline (later referred to as **Synthetic Local Data Augmentation** (**SLDA**) that can generate realistic synthetic instances of hockey sticks within real images, thereby improving a model's ability to detect hockey sticks under adverse conditions.

Objective 2: Develop an **Unity-based 3D Hockey Simulation Tool** that can recreate entire hockey game sequences from positional data, with flexible camera controls and video export. This tool should demonstrate the capability to produce visually realistic hockey scenes that could be used for analytics and for creating additional annotated data (synthetic video) for computer vision model development.

1.3 Overview of Contributions

This thesis makes two major contributions to the field of computer vision in sports analytics, particularly focusing on ice hockey:

1. **Synthetic Local Data Augmentation for Hockey Stick Segmentation:** We introduce a novel data augmentation strategy, termed **SLDA**, to address the challenge

of limited training data for hockey stick detection. In this method, we generate *synthetic stick instances* and blend them into real broadcast hockey images at the pixel level. Unlike global image augmentations, this approach operates at the instance-level data augmentation, effectively "copy-paste augmentation" new stick objects into scenes in a context-aware manner. Key innovations of our approach include: (a) using real stick segmentations (from our collected dataset of hockey stick annotations) as the source for synthetic instances; (b) applying **context-aware transformations** to the inserted sticks, for example, adding motion blur to simulate fast-moving sticks, adjusting brightness/contrast to match lighting, flipping or rotating to create plausible orientations, and scaling the sticks to realistic sizes based on perspective; and (c) ensuring that the synthetic sticks are placed in semantically appropriate locations (e.g., near a player's hands or on the ice surface) to maintain realism. We created a custom annotated dataset of over **4,000 hockey stick instances** in broadcast images to facilitate this study, capturing a wide variety of stick appearances (different stick colors, positions, partial occlusions, etc.). Using SLDA to augment the training data, we trained a stick segmentation model (based on a state-of-the-art instance segmentation framework) and achieved significantly improved performance on stick detection metrics compared to using only real training data. We evaluate the results quantitatively, reporting precision, recall, F1-score, and mAP@50, as well as qualitatively through visual examples. This contribution demonstrates that targeted synthetic augmentation can enhance a model's ability to detect occluded and motion-blurred objects (such as hockey sticks) that are otherwise under-represented in real data. The method and results from this contribution have been peer-reviewed and published as a research study titled **"Synthetic Local Data Augmentation"**, marking a novel addition to the sports vision literature.

2. **Unity-Based 3D Simulation Tool for Hockey Game Reconstruction:** We developed a software tool designed to support future research and development, a Unity-based simulation platform that reconstructs hockey games in 3D from tabular game event data, enabling camera manipulation and visual sequence generation. This tool ingests **(i) time-stamped player/puck coordinates** or **(ii) trajectories of players automatically extracted from video**, and uses these to animate player avatars and the puck within a virtual hockey rink environment. The simulation is visually realistic, including a correctly scaled rink with markings, skatable player models, and a puck model, all moving according to the input data. A major feature of the tool is the ability to control virtual camera parameters: users can place a camera at any location or angle (e.g., mimicking a broadcast camera, an overhead view, or a view from a player's perspective) and adjust zoom levels. They can then render

the sequence from that viewpoint, creating a video of the simulated play. This allows **camera manipulation and multi-view generation** –for instance, the same play can be rendered from the traditional side broadcast angle, from an overhead view, and from on-ice perspectives to create a multi-angle dataset. The tool is designed with extensibility in mind: it could incorporate additional elements such as referees, crowd or arena environment if needed, and supports output of annotations (such as the true positions of each player/puck in the world coordinate system for each video frame). While this simulation tool itself is not a research *algorithm* per se (and thus has not been separately published in an academic venue), it represents an important engineering contribution. It is essentially a **visualization and synthetic data generation framework** for hockey. By documenting and releasing this tool, we provide a platform upon which future research can build, for example, to generate training data for player pose estimation or to validate tracking algorithms by comparing their output on real vs. simulated controlled scenarios. This contribution extends prior work by our lab (such as multiplayer tracking and nomography estimation algorithms) into a full dynamic 3D reconstruction capability. It stands as one of the first open simulation systems for ice hockey analytics, analogous to what has been developed in other sports domains, and fills a gap by enabling **virtual replays and data generation from purely data-driven inputs**.

Collectively, these contributions advance the state-of-the-art (SOTA) in two ways: First, by improving instance-level data augmentation techniques, we show how integrating domain-specific context (such as motion blur for fast objects) can yield better detection models – an insight that can be applied to other sports or domains with similar challenges. Second, by providing a new tool for generating and visualizing hockey game data, we lay the groundwork for a *synergistic pipeline* where outputs from computer vision (e.g., player tracking results, homography transformations) can feed into a simulator to recreate and study games in richer detail. Both contributions are rooted in the idea of leveraging **synthetic data** (whether by augmenting real images or by simulating entire scenes) to overcome real-world data limitations.

1.4 Thesis Organization

The remainder of this thesis is organized as follows. Chapter 2 reviews relevant background and literature to contextualize our work. We begin by examining the challenges unique to computer vision in sports analytics and particularly in hockey, including data scarcity

issues. We then survey existing data augmentation techniques and synthetic data generation approaches, highlighting how our proposed SLDA method differs and improves upon them. Prior research in hockey video analysis, such as player tracking, pose estimation, and puck detection, is also discussed to position our contributions relative to the state of the art. Finally, we cover the role of simulation and game engines in sports vision, which underpins our second contribution.

Chapter 3 delves into the **SLDA** method for hockey stick segmentation. We detail the methodology, including how synthetic stick instances are created and blended into images, the context-aware transformations applied, and the integration into the model training pipeline. We then present experimental results: comparisons of model performance with and without SLDA, analysis on occluded and fast-motion cases, and ablation studies that justify each component of our approach. Visual examples of augmented images and model detection outputs are provided to illustrate qualitative improvements.

Chapter 4 describes the **Unity-based 3D Hockey Simulation Tool**. We outline the system design and components, how players and puck are represented, how input data is parsed, and how the Unity engine is scripted to reproduce the game sequence. We showcase key capabilities such as moving the virtual camera and generating multi-view video. This chapter also demonstrates example use cases: for instance, reconstructing a real game play from NHL, and creating a synthetic dataset of a play from four different camera angles.

Chapter 5 presents a comparative analysis of the SLDA and Unity-based 3D Hockey Simulation Tool contributions. We examine how each method addresses data scarcity and visual complexity from distinct angles: SLDA through targeted object-level augmentation and the simulation tool through full-scene synthesis. This chapter also explores their interoperability and complementary use in model training and synthetic dataset generation, offering a broader view of synthetic data's role in hockey analytics.

Finally, Chapter 6 concludes the thesis with a synthesis of our key contributions and their impact. We highlight how SLDA and the Unity-based 3D Hockey Simulation Tool serve as complementary solutions to the challenges in hockey vision tasks. The chapter also outlines several promising directions for future work, including 3D pose estimation, enhanced rendering realism, and a unified simulation-augmentation pipeline, aiming toward a more comprehensive and intelligent sports analytics system.

Chapter 2

Background and Literature Review

From the problem framing and contributions outlined in Chapter 1, this chapter establishes the technical background and related work that motivate our two contributions. We begin by detailing why hockey video poses unique challenges for computer vision - fast motion, small objects, severe occlusions, and limited labeled data. Then survey synthetic data strategies with an emphasis on object-level augmentation relevant to stick detection. We next review key lines of hockey vision research (player tracking, pose/orientation, puck detection) and conclude with an overview of simulation and game-engine tools for generating controllable, labeled data. Together, these threads clarify the gap our methods aim to fill and lay the groundwork for the targeted stick-level data augmentation (Chapter 3) and the scene-level Unity-based simulator (Chapter 4).

2.1 Sports Analytics and Hockey Video Analysis Challenges

Sports analytics is a broad field encompassing data-driven approaches to understanding and improving sports performance. In recent years, computer vision has become an indispensable tool in sports analytics, as vast amounts of video data are available from games. By automatically analyzing video, one can extract detailed spatiotemporal information that goes far beyond traditional statistics. For instance, in sports such as basketball and soccer, vision-based tracking has been used to study team formations, player movement patterns, and even predict outcomes of plays [14, 31]. Ice hockey stands to similarly benefit from such analysis; however, applying computer vision to hockey video introduces a

number of difficulties.

One fundamental challenge in hockey video analysis is the visual complexity and fast pace of the game. Hockey is played on an ice rink with uniform, low-contrast background (white ice) and with players often clustered tightly, leading to frequent occlusions. Players can occlude each other completely when battling along the boards or in front of the net, and they occlude smaller objects such as the puck and each other's sticks as well. The broadcast camera is typically panning and zooming to follow play, which introduces motion blur and rapid perspective changes. Prakash *et al.* note that tracking players in broadcast hockey footage is extremely challenging due to "occlusions, blurs, camera pan-tilt-zoom effects, and dynamic player movements" [41]. Players can skate and change direction quickly, resulting in sudden non-linear motion that makes it challenging for tracking algorithms to maintain identity consistency [40]. Indeed, even SOTA multi-object tracking systems struggle on hockey footage, often losing track of players when they collide or swapping identities due to abrupt motion changes [40].

Another challenge is the small spatial footprint in terms of pixels, combined with the high speed of key objects such as the puck and sticks. The puck is not only small and moves quickly but also often hidden: it may be against the boards (blending with low-light or shaded areas), behind the goalie, or under players' bodies. A recent study by Salass *et al.* highlights that puck detection in hockey broadcast video is hindered by "the puck's small size, frequent occlusions, motion blur, broadcast artifacts, and scale inconsistencies from zoom" [46, 61]. Essentially, standard object detectors that work on larger, well-separated objects can fail to even perceive the puck in many frames. Hockey sticks, while larger than the puck, present a related challenge, they are thin and elongated, often at sharp angles or partly out of frame, and they can move faster than the players. When a player takes a slap shot, for example, the stick can be a blur. Yet, detecting the stick is important for understanding player actions and context (e.g., differentiating a player holding his stick vs. having dropped it, or identifying stick tackles and penalties) [11].

Compounding these inherent visual challenges is the issue of limited annotated data for training and evaluation. Unlike some other domains of computer vision that have benefited from large public datasets (for example, COCO for everyday objects [34] or ImageNet for general images [17]), sports analytics often relies on proprietary or small-scale datasets. Specifically for hockey, until recently there were no publicly available large datasets for tasks such as player tracking or pose estimation. Researchers have had to create custom datasets from scratch by manually annotating broadcast videos. For instance, Prakash *et al.* (2025) introduced the Hockey Tracking Dataset versions 1 and 2 (HTD-1, HTD-2) with thousands of annotations because "quality datasets" were fundamentally needed to advance hockey tracking algorithms [40]. HTD-1 was a previously developed internal dataset, and

HTD-2 is a newly curated open-source dataset of player bounding boxes and identities from NHL games [40]. The creation of these datasets was a major effort aimed at enabling fair benchmarking of hockey tracking methods. This underscores how data scarcity had been a bottleneck - without such datasets, it was difficult to evaluate or improve algorithms in the hockey domain. Even with HTD-1 and HTD-2 available now, they cover primarily player tracking. For other tasks such as stick or puck detection, publicly annotated datasets remain either non-existent or extremely limited in scope (often confined to a few games or specific scenarios). As a result, models trained on these limited data tend to overfit or fail to generalize to the diversity of real-game situations.

In summary, hockey video analysis is challenged by complex visual dynamics (fast motion, occlusion, small objects) and lack of large labeled datasets. These challenges motivate the pursuit of methods that can make the most of the data we do have, and create new data through augmentation or simulation. The next sections discuss how synthetic data generation and augmentation have emerged as solutions to similar problems in computer vision, and how they can be applied to the sports and hockey context.

2.2 Data Scarcity in Hockey and the Role of Synthetic Data

Data scarcity is a common problem in many specialized domains of computer vision, and hockey analytics is a prime example. Training deep learning models typically requires hundreds of thousands of labeled examples, but obtaining such data in hockey is difficult due to the cost of annotation and the complexity of labeling (especially for objects such as sticks or pucks that are hard even for human annotators to spot in every frame). When only a small dataset is available, models often do not learn robust representations and perform poorly on novel scenarios. This problem has been observed in other contexts as well, for instance, Dwibedi *et al.* (2017) note that a major impediment in deploying object detection models for new environments is "the lack of large annotated datasets," as each new setting (say, detecting objects in a kitchen or a specific sport) would require expensive collection and annotation of many images [21, 44]. Indeed, they point out that it is unlikely to find a large labeled dataset for highly specific tasks (such as our hockey stick segmentation task), and creating one manually is time-consuming and costly [21].

To address limited data, one powerful approach is to generate synthetic data that can supplement real training data. The core idea is to leverage simulations or generative techniques to create images (or videos) for which we automatically know the GT, thus bypassing

manual annotation. If done well, synthetic data can significantly improve model training, either by increasing the volume of training examples or by introducing rare/extreme cases that were missing from the real data [45]. A key advantage of synthetic data is the control it offers: one can systematically vary conditions (lighting, object pose, background clutter, etc.) to make the model more invariant and robust.

In the sports domain, synthetic data generation has started to gain traction [3, 36, 58]. Researchers have built synthetic datasets for scenarios where real data is difficult to obtain. For example, in soccer, some works have constructed entirely simulated soccer matches to generate training data for event recognition algorithms, producing hundreds of minutes of synthetic gameplay video with known ground truth for all events and trajectories (e.g., the "SoccER dataset" dataset of 500 minutes of artificial soccer games for event detection [14]), this demonstrates that simulated sports data can be realistic enough to develop and test analytics algorithms in lieu of or in addition to real data. In autonomous driving, a well-known example is the Virtual KITTI dataset, where Gaidon *et al.* (2016) built a photo-realistic synthetic clone of the KITTI driving benchmark using a game engine. They showed that models (specifically multi-object trackers) performed almost equally well on the synthetic "clone" videos as on the real ones, with a real-to-virtual performance gap of under 0.5% MOTA on average [13]. This indicates that high-fidelity synthetic data can closely approximate real data in terms of the training signal it provides. In some cases, synthetic data has even outperformed real data for training: the team behind the Unity Perception toolkit reported that a model trained mostly on synthetic images slightly outperformed a model trained on the equivalent real images for a 2D object detection task [6]. This striking result underscores the value of synthetic data when real data collection is limited or when we need to cover corner cases extensively.

For hockey, synthetic data can come in two main forms: **(1)** synthetic images created by augmenting real images (e.g., pasting new objects or altering existing ones), or **(2)** fully synthetic rendered images from a simulation (which we discuss later in Section 2.6). Both approaches can expand the training distribution. The role of synthetic data in our context is essentially to provide the model with examples that are otherwise rare, for instance, an image where a hockey stick is heavily motion-blurred or only partially visible behind a player. If the model never or rarely sees such an example in real data, it will likely fail on it during testing; but if we can synthesize a few dozen such instances and include them in training, the model may learn to handle that scenario.

Note that synthetic data is not a magic bullet, if not generated carefully, such data could introduce unrealistic artifacts or biases that actually hurt model performance. The goal is to make synthetic examples as realistic and relevant as possible, so that the model treats them as useful training samples rather than outliers. A concept known as domain

randomization suggests that even if synthetic data is not perfectly realistic, if one introduces enough variation in the synthetic images, the model can learn the underlying task in a way that generalizes to real images. Tobin *et al.*. showed this for robotic object detection by randomizing colors, textures, and lighting in a simulator, the model learned to focus on essential features and ignored the unrealistic aspects, ultimately transferring to real-world images [54, 55]. In hockey, we similarly want to randomize certain parameters (such as stick position, orientation, motion blur amount, etc.) in synthetic generation to cover a wide range of appearances.

In summary, synthetic data serves as a remedy for data scarcity in hockey vision tasks. Whether by augmenting existing real images or by generating new images from simulation, it provides additional "free" training data that can enrich the model's learning. In the following section, we delve into specific techniques for object-level data augmentation, which is one category of synthetic data generation particularly relevant to our first contribution (SLDA). We will see how past work has inserted objects into images to boost detectors, and how we build upon those ideas.

2.3 Object-Level Data Augmentation Techniques

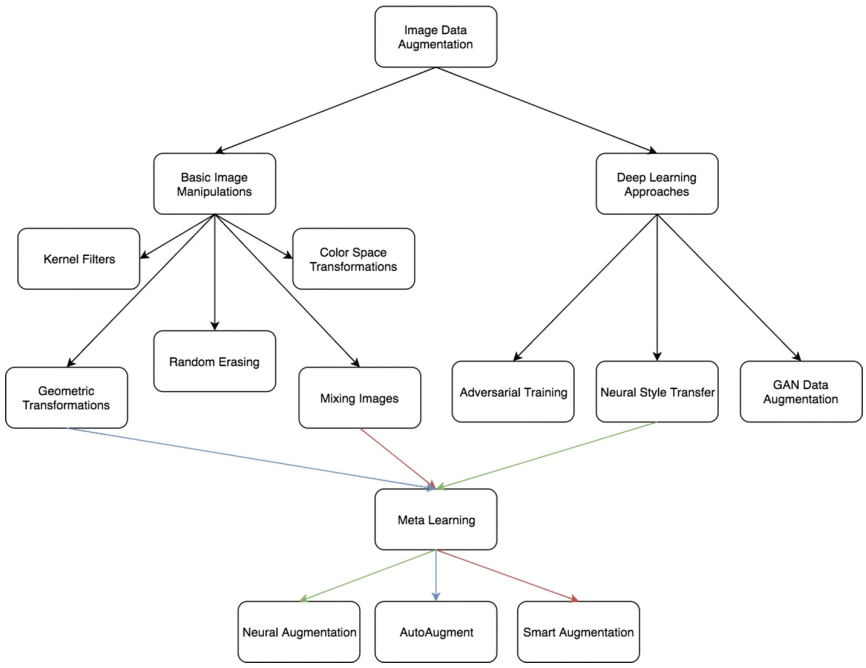


Figure 2.1: A taxonomy of image data augmentation techniques, divided into basic manipulations (left branch) and deep learning approaches (right branch). This diagram organizes augmentation methods (geometric transforms, color adjustments, mixing images, adversarial training, style transfer, GAN-based generation, etc.) and highlights how meta-learning (colored lines) connects to each category. [50]

Data augmentation in computer vision refers to the practice of applying transformations to existing images to create new training samples. Figure 2.1 illustrates a taxonomy of common image augmentation techniques [5, 50]. Common global augmentations include geometric transforms (rotation, scaling, cropping, flipping) and photometric adjustments (color jittering, brightness/contrast shifts, blurring). Such augmentations help encode invariances (e.g., a rotated object is still the same object) and can improve model generalization [24]. However, these traditional augmentations treat the entire image as a whole; they do not introduce new object instances that were not already in the image.

For tasks such as object detection and segmentation, there has been growing interest in object-level or instance-level augmentation, that is, augmentations which *add or modify objects* within images rather than just altering the image globally. One straightforward form

of instance-level augmentation is the "copy-paste" strategy: cut out an object from one image and paste it into another. This effectively increases the number of object instances in the training set and can create novel combinations of objects and backgrounds.

A seminal work in this area is by Dwibedi *et al.* (2017), titled "Cut, Paste and Learn", which demonstrated how surprisingly effective simple synthetic composition can be for instance detection [21]. They automatically extracted object masks from some source images and pasted them onto random background images. A naïve copy-paste might produce visible artifacts (edges or lighting inconsistencies), but the authors addressed this by methods to make the model ignore these artifacts (for example, by blending or by using an autoencoder to clean the pasted images). Their results were impressive: using synthetic composite images, they outperformed earlier more complex synthesis approaches, and notably, they showed that a detector trained on a combination of synthetic data and a small fraction (10%) of real data could exceed the performance of a detector trained on 100% real data [21, 39]. In other words, 10% real + synthetic > 100% real, illustrating the power of augmentation in low-data regimes.

Following this, several other works refined the copy-paste augmentation idea. In 2021, Ghiasi *et al.* presented "Simple Copy-Paste" for instance segmentation, a method which emphasized that even *randomly* pasted objects can yield significant gains in detection accuracy [24]. They found that taking two images, cutting objects from one and pasting onto the other at random positions, improved performance across a variety of settings [18]. When combined with large-scale training, copy-paste yielded notable improvements in data efficiency: for example, they reported a twofold improvement in data-efficiency on the COCO dataset and a +10 AP gain in a low-data regime using only 10% of COCO training data [24]. These improvements carried through to SOTA results on COCO when using copy-paste alongside advanced architectures [24]. The success of this method suggests that even without ensuring perfect context for pasted objects, the sheer increase in instance count and diversity helps the model learn better. Copy-Paste is easy to implement and introduces no overhead during model inference, making it an attractive augmentation.

Despite the "simple is effective" message, other researchers explored adding some *context awareness* to object-level augmentation. One example is InstaBoost by Fang *et al.* (ICCV 2019), which aimed to place objects in more realistic locations when pasting. The idea was to use the existing dataset's images to learn where an object might plausibly appear. InstaBoost uses a probability map guided by local appearance to decide the placement of a new object cut-out [22]. Essentially, it looks at the destination image and finds regions where the object's shape/color might fit (for instance, you would not paste a player on top of another player, you would paste them on the ice next to other players). By doing so, it avoids absurd placements and respects some context. InstaBoost also jitters the

objects (random small rotations, flips, etc.) to augment them further [50]. This guided approach yielded improvements on instance segmentation benchmarks, for example, they reported boosting Mask R-CNN's performance on COCO by about 2.2 mAP (from 35.7 to 37.9) without changing the model, just by using their augmentation strategy [22]. This demonstrates that respecting context (even in a simple form) during copy-paste can further enhance results beyond random paste.

Beyond copy-paste of 2D cutouts, some augmentation methods generate synthetic objects via rendering or generative models [2, 52]. For example, one could render a 3D model of a hockey stick onto an image (if a 3D model is available), or use a GAN to generate new examples of sticks. These are more complex and ensuring realism can be difficult, so our work focuses on 2D instance augmentation which is more controllable using real data cutouts.

In the context of hockey and our problem, object-level augmentation translates to *augmenting hockey scenes with additional or modified hockey objects*. Our focus is on hockey sticks as the object of interest [11]. Traditional augmentation might slightly help (e.g., flipping an entire image might create a left-right swap of a scene, which for hockey does not change much except which side of the rink the play is on), but it does not create new sticks or new occlusion scenarios. By using object-level augmentation, we can insert synthetic hockey sticks into images to create scenarios that were missing or underrepresented in the original data. For instance, if our real dataset had few examples of sticks on the ice (maybe most players held their sticks waist-high in the images), we could paste some sticks lying flat on the ice to teach the model that sticks can also appear in that orientation.

However, directly applying copy-paste augmentation to hockey sticks needs to be done thoughtfully. A pasted stick should look natural: if it is supposed to be a fast-moving stick, perhaps it should be a bit blurred; if it is in the distance, it should be scaled down appropriately; if it is on the ice, maybe a slight reflection or shading might make it more realistic, etc. A potential issue is occlusion ordering, for example, a stick should appear behind a player if the player is closer to the camera, not pasted on top of them. [64] Naïve copy-paste could end up pasting a stick over a player's body where it clearly should have been occluded. Addressing these details moves us into the next section, where we discuss context-aware augmentation specifically aimed at handling occlusion and motion cues, which is directly relevant to how we designed the method.

2.4 Context-Aware Augmentation for Occlusion and Motion Blur

One of the key contributions of our work is making object augmentation *context-aware* for the specific issues of occlusion and motion in hockey imagery. The principle of context-aware augmentation is to incorporate knowledge of the scene or object dynamics so that synthetic modifications appear as plausible as possible to both human observers and the learning algorithm.

In hockey images, sticks are often partially occluded by players. If we want to synthesize a new stick instance in an image, we should consider whether it should be partially hidden behind a player or another object. A simple approach to this is using layering and depth ordering[57, 62]. For example, if we know roughly the depth order (say, we want the stick to be on the ice behind a player's skates), we can paste the stick and then *re-paste* an excerpt of the ice or the occluding object to cover parts of the stick, simulating occlusion. Another approach is to leverage existing masks, if the dataset has player segmentation masks, we could paste a stick and then overlay the player mask on top of it to naturally occlude the portion behind the player [26]. In our augmentation pipeline, we indeed consider such strategies: when inserting a synthetic stick, we check if its intended position intersects with a player region; if so, we can make the stick appear behind the player by using the player's mask.

Motion blur is a significant factor in hockey images due to fast motion. A stick swung rapidly can appear as a semi-transparent blur rather than a sharp object. Models trained only on sharp images might not recognize the blurred version as the same object. Therefore, adding motion blur to synthetic objects can teach the model to handle it. In our context, we apply directional Gaussian blur to the pasted stick instance to mimic the effect of motion. The direction and magnitude of blur can be randomly chosen or informed by context (for instance, horizontal blur if the stick is moving horizontally across the image). This concept is akin to what some augmentation techniques do for robustness, for example, there are augmentation methods for classification that add blur to improve robustness to camera shake [28, 50], and for segmentation, researchers have synthetically blurred regions to improve segmentation under motion [1]. By blurring synthetic sticks, we aim to produce training examples where the model sees a faint streak and learns that it could still be a stick.

Hockey broadcast videos are typically from a perspective view (somewhere in the stands, looking slightly down at the ice). Therefore, objects farther from the camera appear smaller. When pasting a stick, its pixel size should be consistent with its intended location

in the scene, a stick near the far boards should appear shorter (fewer pixels) than a stick near the camera. We address this by scaling synthetic stick instances before insertion [51]. Our augmentation algorithm takes into account the sizes of existing sticks already present in the image, allowing it to insert synthetic sticks at appropriate scales that match the scene context. This ensures that the synthetic sticks do not look out-of-place due to incorrect sizing.

Context-aware augmentation also involves matching the appearance of inserted objects to the scene. For example, if the overall image has a particular color tone (e.g., bluish from arena lighting), a poorly matched stick might look out of place. To maintain realism, we use a curated dataset of segmented sticks collected from real hockey footage, this dataset already includes a diverse range of stick types, colors, and tape styles. When inserting these sticks, we preserve that diversity by sampling from this varied set, which naturally introduces appearance variation. This helps the model avoid overfitting to any single stick appearance. Minor adjustments (e.g., to brightness or contrast) may still be applied to improve blending, but the primary source of visual diversity comes from the variety of real-world examples in the insertion pool [43].

By making augmentation context-aware in these ways, we are essentially extending the idea of InstaBoost (which focused on location context) to *dynamic context* relevant in hockey: motion and occlusion. Our SLDA method, which will be detailed in Chapter 3, builds upon the copy-paste foundation but adds these layers of realism. Early results from our study show that these details matter, for example, a model trained with sticks pasted *without* blur performed worse at detecting blurred sticks than a model trained with blurred synthetic sticks included. Similarly, ensuring some synthetic sticks are partially occluded during training helped the model not to miss those in real images.

Context-aware augmentation has parallels in other domains: for instance, in autonomous driving, one might insert synthetic cars with appropriate motion blur to simulate different speeds, or in medical imaging, insert synthetic tumors with shape variations that match surrounding tissue context [27][32]. Our work can be seen as bringing such ideas into the sports analytics domain.

In conclusion, object-level augmentation techniques such as copy-paste provide a framework to boost training data, and by incorporating domain-specific context, in our case, handling occlusions and motion blur for hockey sticks, we can further refine the synthetic data to be highly effective. This addresses the first part of our thesis (improving hockey stick detection), which sits at the intersection of data augmentation and sports vision. Next, we shift focus to related works in hockey analytics that tackle the problems of tracking players and pucks, as well as the use of simulation in sports, to provide background for

the second part of our thesis (the simulation tool).

2.5 Player Tracking, Pose Estimation, and Puck Detection in Hockey

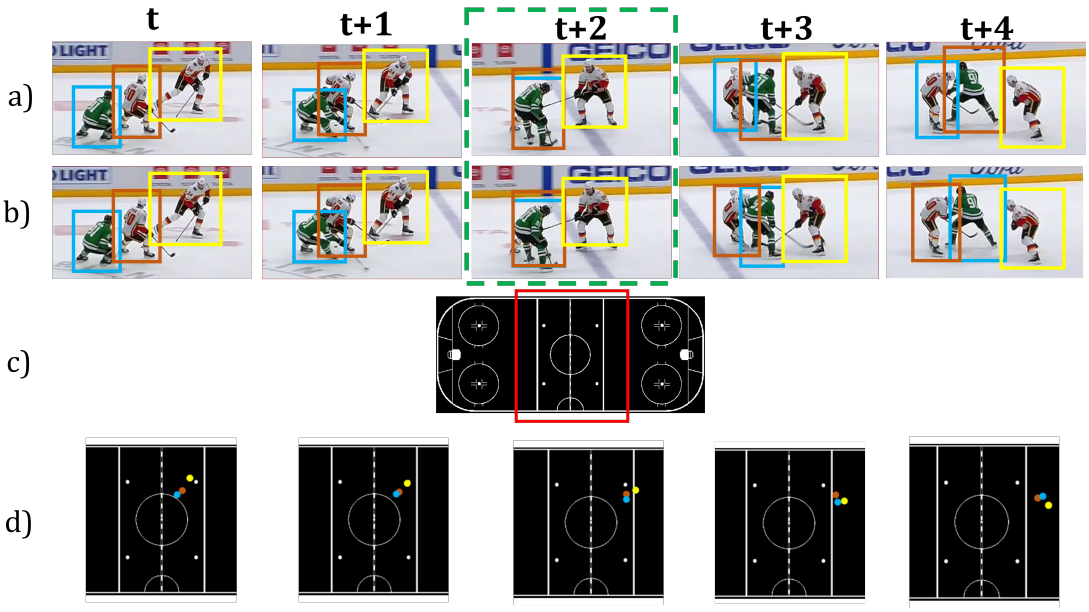


Figure 2.2: Occlusion-aware player tracking in ice hockey. In scenario **(a)**, a standard tracker fails when two players overlap, causing an ID switch (player identities swap colors). In **(b)**, the approach of Prakash *et al.* (2024) maintains consistent tracking before, during, and after the occlusion by using an overhead rink projection. **(c)** and **(d)** depict the overhead rink template and the mapped footpoint positions of players; even when players overlap in the broadcast view, their projected positions (red and blue dots) are distinct on the rink, enabling the tracker to keep their identities separate. (Prakash *et al.*, 2024) [40]

The computer vision community has recently seen a surge of interest in analyzing hockey games, spurred by the unique challenges and the availability of some new datasets. We highlight here a few areas of related work: multi-player tracking, player pose estimation/identification, and puck detection, each of which provides context or building blocks for what a comprehensive hockey analysis system might involve.

Multi-Player Tracking: This is the task of detecting and consistently tracking all players (and sometimes officials) on the ice through video frames, maintaining their identities over time (often termed multi-object tracking or MOT) [16]. Early hockey tracking efforts (circa 2004–2010) employed traditional multi-object tracking (MOT) pipelines. Okuma et al. pioneered hockey player tracking using an AdaBoost detector combined with a particle filter to predict movements [38]. Later, Cai *et al.* improved this by using mean-shift clustering and integrating rink homography (projecting player positions to a 2D rink coordinate system) in the particle filter [10]. These classical methods, however, struggled with the sport's complexity – identity switches were common when players overlapped, and trackers would lose players during fast turns or occlusion by the boards [40]. In the last few years, deep learning has significantly advanced player MOT in hockey. Vats *et al.* [59] evaluated several modern trackers (e.g., SORT, DeepSORT, and graph-based trackers) on NHL footage. They achieved SOTA results using a graph neural network with message-passing (MPN) to associate detections, establishing the first public benchmark for hockey player tracking [7]. However, even their approach had limitations under heavy occlusion and similar-looking players. A novel solution by Prakash *et al.* [40] explicitly tackled these issues by incorporating a homography projection of player locations. In their method, each player's foot position is mapped onto a standard overhead rink template, creating a bird's-eye view where overlapping players become separable by spatial distance [40]. This overhead position is used as an additional feature in a graph tracking network, leading to more consistent track identities through occlusions. Figure 2.2 illustrates this occlusion-aware tracking approach: in a challenging sequence where two players collide and separate, a conventional tracker suffers an identity swap, whereas the homography-informed tracker maintains correct identities by leveraging the overhead context. By disentangling players in a top-down coordinate frame, the tracker can "see" through occlusions that confuse the camera view.

This concept is relevant to our simulation tool: if one can map broadcast detections to rink coordinates, those coordinates could feed directly into a simulation (since the simulation operates in rink coordinates as well) [29]. In fact, one long-term vision is to take outputs from works such as Prakash's tracker and *recreate* the game in Unity in real-time, effectively achieving automated "camera-to-simulation" conversion of a broadcast.

Player Pose Estimation and Identification: Besides tracking players as boxes or points, another level of analysis is estimating players' body pose (joint positions) or identifying who the player is (by jersey number or other features). In hockey, pose estimation is extremely challenging due to the players wearing bulky equipment and being seen from a distance. Standard pose estimators (such as OpenPose [12] or COCO [34] pre-trained models) often falter on hockey players because the limbs can be occluded or the poses are

very dynamic (players frequently bent over skating, etc.). Nonetheless, research is moving in this direction. For example, research has investigated the use of 2D pose estimation specifically on hockey by fine-tuning on hockey images or by using multi-view footage [8] (e.g., a multi-camera setup to get better angles). Balaji [4] conducted a project on "End-to-End Pose Estimation from Monocular Ice-Hockey Videos," which indicates the interest in extracting player stick and limb positions from single-camera feeds. While detailed results of that project are not published widely, it involved developing robust pose models for hockey using course project data. In general, pose estimation could benefit from synthetic data too, one could imagine generating synthetic hockey players in various poses (via simulation or augmentation) to train a pose model, given the paucity of annotated hockey pose data.

More recently, Houshmand *et al.* [47] introduced HockeyOrient, a dataset of 9,700 annotated video frames from the Swedish Hockey League focusing on player orientation classification. Rather than full 2D pose, this work targets the direction a player is facing (e.g., left, right, up-ice, down-ice), which is a key tactical variable. They report that using pose cues (such as the torso and head direction) can classify orientation with high accuracy, enabling new analytics such as how players position themselves relative to play. The HockeyOrient dataset fills a gap by providing pose labels in a hockey context, and it can support training specialized pose estimators or validating multi-person pose models on hockey data. Overall, player pose/orientation estimation in hockey is still an emerging area. It shows promise for enriching player tracking (since knowing which way a player is facing can improve predictions of play development) and for downstream tasks such as action recognition or assessing checking mechanics for safety. Future work will likely leverage existing human pose frameworks, fine-tuned with hockey-specific data, and possibly multi-view camera setups (e.g., top-down cameras that capture body poses more clearly than side-angle broadcasts).

Another related problem is player identification, i.e., determining the identity of a player (which usually means recognizing jersey numbers or using facial features if visible) [35]. This often piggybacks on tracking: once you track a player through a sequence, you can propagate their ID. Prakash's tracking system, for instance, maintained identities but did not necessarily match them to jersey numbers, that would be an additional layer (some recent hockey analytics works use OCR on jersey numbers or manual tagging to label tracks with player names) [60]. While this thesis does not directly tackle player identification, the developed simulation simulation tool could incorporate it by reading which player (by ID) is at which trajectory, and then maybe rendering them with a unique jersey to visually distinguish them.

Puck Detection and Trajectory Estimation: The puck is a crucial object and has been the subject of dedicated research [61]. We already mentioned the difficulty of puck detection. A recent approach by Salass *et al.* (2025) called PLUCC (Puck Localization Using Contextual Cues) [46] introduced an ingenious way to detect the puck by leveraging *context from players*. They observed that players' behavior provides strong signals, players tend to face or stick-handle towards the puck. So, PLUCC uses a contextual encoder that processes player orientations and positions, effectively predicting likely puck regions from how players are arranged. This context is fused with a standard image feature encoder, and a decoding network then outputs puck detections. By incorporating these contextual cues, their model achieved SOTA performance on a hockey puck detection dataset, with about a 12.2% average precision improvement over previous baselines. This result is significant: it shows that understanding the game (where players are looking, how they are positioned) can substantially aid in detecting something as small as the puck. It is a reminder that purely appearance-based detection (looking for a tiny black disk) might not be sufficient, a higher level context should be employed. In our work, while we focus on sticks rather than the puck, similar logic applies: a hockey stick's location is not independent of the player; normally, a player's stick is in their hands. Such contextual expectations could be used in future to guide stick detection (e.g., one could enforce that a detected stick should be proximate to a player's hand position as estimated by pose). Although we did not explicitly encode that in our current model, our simulation tool inherently respects such context (since it places sticks with players). Moreover, if one had a simulation that outputs both player and puck trajectories, it could generate synthetic training data for a model such as PLUCC by providing rich examples of the relationship between player orientation and puck location.

Integration of Multiple Cues: Ultimately, solving hockey understanding may require combining these tasks, tracking players, estimating poses, detecting sticks and pucks, into one coherent system [40][46][4]. Prior works have taken steps in that direction. For example, some approaches to event detection in sports consider both the trajectories of players and the occurrence of certain visual cues (such as a stick swing for a shot, or a goalie in butterfly pose for a save) [31][3][14]. The output of tracking (who is where) can feed into higher-level reasoning about the game. Our simulation tool can be viewed as a means to unify these: if one can take the outputs of player tracking (from broadcast video) and outputs of puck tracking, one could feed them into the simulation to reconstruct the play. Additionally, a simulation environment could allow algorithms to be tested end-to-end, e.g., generate a synthetic video from known data, run a tracking algorithm on it, and compare the estimated trajectories to the ground-truth trajectories that were fed into the simulator, thereby quantitatively evaluating the algorithm's performance in a controlled

scenario [14][23][6][13].

In summary, the literature on hockey computer vision has begun to address the various pieces: tracking with homography-assisted models to mitigate occlusion, pose estimation and identification for finer player details, and contextual detection for small objects such as the puck. This thesis builds on this body of work in two ways. First, our augmentation approach for sticks can be seen as complementary to these efforts, while others improve tracking or puck finding, we improve the detection of the stick, which is another important piece (for example, knowing the stick position could help determine if a player is about to pass or shoot). Second, our simulation tool is intended to complement these algorithms: it is an enabling technology that could visualize their outputs or generate inputs for them. For instance, the tool could generate synthetic scenarios to train a multi-object tracker (by rendering many simulated games and providing ground truth positions), or it could take the output of a multi-object tracker and create a visual replay for qualitative analysis.

2.6 Simulation Tools and Synthetic Data Generation for Sports Analytics



Figure 2.3: Example of a sports simulation environment – a rendered game scenario from the Google Research Football simulator. This advanced 3D engine models players, ball physics, and game rules of soccer, providing a rich synthetic dataset for developing and evaluating computer vision and RL algorithms [33].(Kurach *et al.*, 2020)

Simulation and the use of game engines have become increasingly popular in computer vision as a source of unlimited, labeled data. In sports analytics, simulation serves a dual purpose: visualization (recreating scenarios for analysis or fan engagement) and data generation (producing annotated data to train or evaluate algorithms).

General-purpose game engines such as Unity and Unreal Engine are powerful platforms for creating simulated environments. They offer high-quality rendering, physics, and the ability to script scenarios [30, 42]. Recognizing their value for AI, the Unity team even released the Unity Perception package to streamline synthetic data creation for vision tasks [6]. With Unity Perception, users can set up scenes and randomize object placements, materials, lighting, etc., and automatically capture images with ground truth annotations (bounding boxes, segmentation masks, etc.). Borkman *et al.* (2021) noted that Unity Perception can produce perfectly annotated images for detection, and models trained on mostly synthetic data from Unity performed well, outperforming models trained on only real data in their experiments [6]. This underscores that a well-designed simulation can generate data of sufficient quality for serious model training.

A prominent example is the Google Research Football Environment introduced by Kurach *et al.* (2020) [25, 33]. This is a physics-based 3D soccer simulation platform where AI agents can play full matches in a realistic virtual stadium. Originally developed for reinforcement learning research, the Football Environment is open-source and supports multi-agent play, varied difficulty scenarios, and full control over game state. Figure 2.3 shows a screenshot of this simulation: the virtual players and ball behave like in a real soccer match, complete with rules (kickoffs, fouls, corner kicks, etc.) [25, 33]. Such an environment not only accelerates machine learning research (by providing a controllable, reproducible setting) but also serves as a synthetic data generator, one can record thousands of simulated games to train a vision model to recognize players, track the ball, or detect events, then transfer that knowledge to real-world footage [25, 33].

Other simulation is also notable in soccer, researchers have built simulators to generate data for camera calibration and broadcast understanding. Chen and Little (2019) developed a synthetic data approach for sports camera calibration, where they generated thousands of images of soccer fields with varying camera angles using a graphics engine, to train a model that can estimate camera pose from a single real image [13, 53]. Their synthetic data approach achieved SOTA accuracy on real soccer and volleyball datasets for camera calibration [13]. The success here came from simulating the exact elements needed (the field with markings) and randomizing camera parameters, an approach we also adopt by simulating many camera viewpoints in our Unity tool.

In ice hockey, to our knowledge, there has not been a widely published 3D simulation

tool or synthetic dataset prior to our work (aside from possibly internal tools used by companies or unreleased projects). Our Unity-based 3D Hockey Simulation Tool aims to fill this gap. We leverage the Unity engine's capabilities to create a realistic rink environment: this includes the rink dimensions to NHL specifications, the markings (lines, circles), goals, etc., and placeholders for players and the puck. Since appearance realism is less crucial for our current goals (we are not, for example, simulating cloth physics of jerseys), we focus on *positional accuracy* and *camera realism*. Positional accuracy means that if the input data says a player is at coordinate (x, y) on the rink at time t, our simulation will place the player exactly there at that time. Camera realism means when we simulate the view, the output should mimic what a real camera might see (perspective projection, FoV, some amount of motion blur if the camera is moving, etc.) [48, 49].

One technical aspect is synchronizing the simulation timeline with the input data's timeline. We ensure that the Unity simulation runs on the same temporal resolution (e.g., if we have data at 30 frames per second, the simulation will update positions accordingly each 1/30th of a second). Unity's scripting API allows us to feed position data frame by frame.

The camera manipulation feature of our tool is worth expanding on. In Unity, one can create virtual cameras and specify their transforms (position, rotation) and properties (field of view, depth of field, etc.). Our tool provides a user interface (or configuration file) to set a camera angle. For example, one can choose a classic broadcast side view (elevated at center ice, looking down), or an overhead view (bird's eye), or even a view from a player's helmet (which could be used to simulate first-person footage). Because Unity can render the scene from any of these cameras, we can output video from multiple viewpoints. If needed, the tool can output annotation data as well: Unity can provide the exact 2D screen coordinates of each object (players/puck) for each frame, effectively giving perfect detection/tracking labels for the synthetic video [6, 19].

Applications of Simulation: Having such a simulation tool opens up many possibilities:

- *Action recognition and strategy analysis:* By generating multi-view videos of the same play, one can train models that take multiple camera feeds as input (something not possible with a single real broadcast feed). Or one can simulate many variations of a scenario (e.g., a power-play setup with players in different arrangements) to train a model to classify strategies or set plays.

- *Data augmentation for tracking:* Synthetic videos could be added to a tracker's training set. For instance, one could simulate some common but challenging scenarios

(such as players crowded in front of the net) to augment a tracking algorithm's training, ensuring it learns to handle those better. This is akin to how Virtual KITTI is used to augment real KITTI for tracking [9, 23].

- *Validating algorithms:* If we have a new algorithm for, say, identifying when a player is offsides from video, we could test it on simulated scenarios where we deliberately violate or adhere to the offsides rule and see if the algorithm catches it. Since in simulation we know the ground truth of offsides (we scripted it), we can quantitatively measure performance.

- *Legacy and reproducibility:* By releasing a simulation tool, future researchers can reproduce our synthetic data or generate their own. It serves as a common framework where others can plug in their data or algorithms. Prior work in the Vision and Image Processing Lab (VIP Lab) emphasized that this tool is a "living legacy", meaning it will remain useful after the thesis, allowing others in the lab or the broader community to build on it.

Simulation tools benefit greatly from community contributions. For example, if someone later adds improved skating animations to our Unity project, the synthetic data's fidelity improves and could benefit everyone using it. The initial version we present focuses on core functionality (position-accurate replay, camera control, etc.), leaving visual enhancements as future improvements.

In terms of related work, perhaps the closest analogues to our hockey simulator are tools used in broadcast sports. Companies such as Sportlogiq or Second Spectrum (which provide analytics for sports broadcasts) often create visualizations of games in simplified 3D form for broadcasters. While those are proprietary, they indicate that the concept is feasible: given player tracking data, one can render a representation of the game. Our approach differs in that we aim for more *general* use (not just visualization but also data generation for computer vision tasks), and we integrate the output with research needs (e.g., the ability to export ground-truth labels, to randomize cameras, etc.).

Finally, a note on limitations: a simulation is only as good as the input data and the modeling. If the input player trajectories are noisy or if important aspects such as player orientation are missing, the simulation might look slightly off (players sliding sideways, etc., if orientation isn't known). We mitigate some of this by simple interpolation for orientation (e.g., face the direction of movement, or face the puck). Incorporating more human-like animations triggered by events (such as a shooting animation when a shot event is in the data) would make the simulation more realistic and informative (one could then

even detect those events from the synthetic video to validate event-detection algorithms). This is planned as future work. Despite these simplifications, the simulation still provides accurate spatial and temporal representation of the play, which is the critical part for generating data for algorithms.

2.7 Summary of Literature

In this chapter, we surveyed the landscape of research and tools related to our thesis. We first discussed the unique challenges that make hockey video analysis difficult, fast movements, occlusions, small objects, and scarcity of labeled data. These challenges set the stage for why our contributions are needed. We then explored how synthetic data and augmentation have emerged as solutions to data scarcity in various domains. Notably, simple copy-paste augmentation has proven to yield significant performance gains for detection and segmentation models, especially in low-data settings [21], [24]. Building on this, context-aware augmentation techniques such as InstaBoost show that respecting scene context can further improve realism and effectiveness [22]. These insights directly informed our Synthetic Local Data Augmentation (SLDA) approach, where we apply object-level augmentation tailored to the hockey context (introducing synthetic sticks with appropriate blur, occlusion, and scaling).

We also reviewed key related works in hockey vision: multi-player tracking leveraging homography for occlusion handling [40], pose estimation and identification efforts for players, and contextual puck detection that uses player cues to find the puck [46]. These works illustrate the trend of injecting domain knowledge (geometry, context) into computer vision models to tackle hockey's challenges, a philosophy our work shares. For example, our augmentation encodes knowledge of motion blur and typical stick locations, and our simulation encodes rink geometry and camera perspectives.

Finally, we delved into simulation tools and synthetic data generation, highlighting successes such as Virtual KITTI in driving (minimal reality gap [23]) and Unity Perception's applicability in vision model training [6]. We argued for the potential of a hockey simulation tool to both visualize plays and generate new data, noting analogies in other sports and early uses in broadcast analytics. The literature (and industry practices) strongly suggest that simulation is a growing area in sports technology, our contribution in this space is timely and novel for hockey.

In conclusion, the background reviewed here shows that our thesis builds upon a solid foundation of research: we adopt proven techniques such as instance augmentation and

simulation, adapt them to the domain of hockey with novel twists (context-aware stick augmentation, a first-of-its-kind open hockey simulator), and integrate ideas from related works (homography usage, contextual cues) to maximize the impact. The subsequent chapters will detail how we implemented these contributions and present empirical results that demonstrate their effectiveness in advancing hockey video analytics.

The literature review in Section 2.7 spans augmentation, tracking, and simulation. Table 2.1 condenses that discussion into a single comparative view so readers can (i) see which sub-tasks each study targets, (ii) note the datasets and headline improvements, and (iii) spot the open problems, particularly the lack of blur/occlusion-aware stick augmentation and an open 3-D hockey simulator.

Table 2.1: Summary of representative augmentation and simulation papers across domains. Reported gains are in limited-data settings unless otherwise noted.

Paper (year)	Core task / domain	Public dataset(s)	Key idea / technique	Headline gain*	Outstanding gap
Dwibedi *et al.* 2017 – *Cut, Paste & Learn*	Instance detection	Pascal VOC 2007 (targets) + COCO (bkgd)	Simple object-mask copy-paste compositions	**+11 mAP** with 10% real data (beats 100%-real baseline)	No scene context or motion cues
Ghiasi *et al.* 2021 – *Simple Copy-Paste*	Instance segmentation	COCO	Random object swap between image pairs	**+10 AP** in 10%-data regime; **+2 AP** full-data	Ignores scene semantics
Fang *et al.* 2019 – *InstaBoost*	Instance segmentation	COCO	Probability-guided contextual paste + jitter	**+2.2 mAP** over Mask R-CNN baseline	No motion / blur modelling
Prakash *et al.* 2025 – *Broadcast Is All You Need*	Multi-player tracking (hockey)	HTD-2	Homography-aware graph matching	**10**× fewer ID-switches; **+32 IDF1** vs SORT baseline	Needs stick / puck context
Salass *et al.* 2025 – *PLUCC*	Puck detection (hockey)	NHL clips	Player-context encoder + image CNN	**+12.2 AP** over prior SOTA	Stick localisation unresolved
Gaidon *et al.* 2016 – *Virtual KITTI*	MOT & detection (driving)	Virtual KITTI clone	Photo-realistic scene replication	**<0.5%** MOTA gap real ↔ synthetic	Sports domain unexplored
Borkman *et al.* 2021 – *Unity Perception*	2-D object detection	Synthetic household scenes	Large-scale parametric randomisation	Synthetic-only model within **0.6 mAP** of real-only	No sports assets
Chen & Little 2019 – *Synthetic fields*	Camera calibration (soccer)	10k rendered pitch images	Engine-rendered field lines + CNN regressor	Reprojection error ↓ **25%** vs prior best	Static field lines only

Chapter 3

Synthetic Local Data Augmentation (SLDA)

Building on the challenges outlined in Chapter 2, this chapter introduces Synthetic Local Data Augmentation (SLDA), an object-level augmentation method designed to improve hockey stick segmentation under motion blur and limited resolution. The approach operationalizes the synthetic data principles introduced earlier by (i) generating stick instances with controlled, object-specific transformations, (ii) adjusting their size and resolution to match the scene context, and (iii) inserting them at plausible on-ice locations. We then evaluate the method's impact on segmentation accuracy and robustness to blur.

3.1 Introduction

Modern object segmentation models are crucial in sports analytics, particularly in dynamic sports such as hockey, where fast-paced action often results in blurred imagery (e.g., motion-blurred hockey sticks). Precise recognition of hockey stick positions is key for analyzing player actions in real time, as it informs a player's current action and aids in unique player identification (through details such as handedness and stick design). However, hockey game footage presents crowded scenes with multiple players and overlapping equipment, where traditional object detection methods (e.g., bounding boxes) struggle. The challenge is exacerbated by motion blur: hockey sticks are relatively small and low-resolution in broadcast frames, making them particularly susceptible to blur during rapid movements. This motion blur distorts visual cues needed for effective segmentation, rendering sticks nearly invisible in extreme cases.

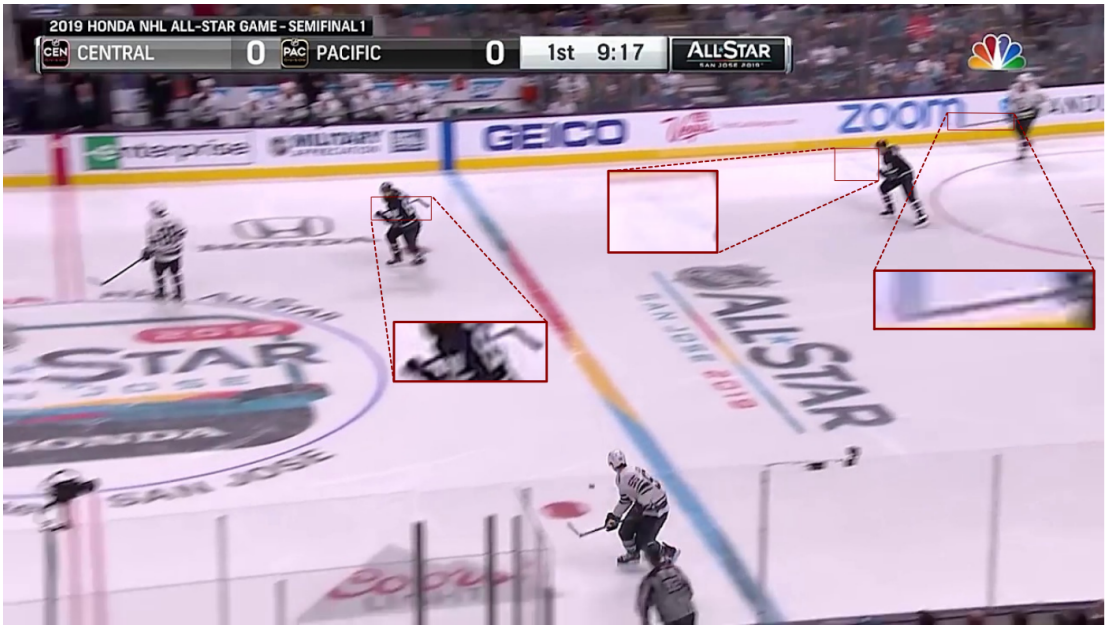


Figure 3.1: Example of an image used in hockey stick segmentation, highlighting the challenges posed by the small size of a hockey stick (∼600 pixels, or 0.06% of the image). Challenge cases (red boxes) show that motion blur on a low-resolution stick can make it nearly invisible.

Compounding these difficulties is the scarcity of specialized data. Hockey sticks are uncommon objects not typically included in large public image datasets, due to the difficulty of capturing and annotating these fast-moving, thin objects. Given the limited training data for hockey stick segmentation, data augmentation naturally emerges as a crucial technique to increase data diversity and volume. Yet, widely used image-level augmentation techniques (e.g., erasing parts of images or mixing images) that have proven effective in general settings are not specifically designed for instance segmentation challenges. Methods that randomly alter or blend image features can disrupt precise object contours and contextual relationships, which are critical for segmenting specific small objects such as hockey sticks.

Among specialised instance augmentation techniques, Simple Copy-Paste [24] and ObjectAug [63] each have limitations. Copy-Paste introduces new object contexts but does not adjust the inserted object's size and resolution to match the scene, which is crucial for low-resolution objects. ObjectAug perturbs existing scene objects, but fails to introduce new object instances and struggles to control distortion levels because it uses objects

already present (with their inherent, varying distortions). In summary, existing methods either ignore scale/resolution mismatches (Copy-Paste) or lack diversity and control (ObjectAug) for our task.

Recognising these gaps, we developed *Synthetic Local Data Augmentation* as a novel object-level augmentation technique tailored for hockey stick segmentation. SLDA operates at the object level: it selectively applies traditional augmentation transformations (scaling, rotation, blurring, motion blur, etc.) directly to individual target objects (the hockey sticks), rather than the entire image. By altering only the relevant object, SLDA maintains the context and integrity of the surrounding scene. Transformations can be precisely customized to enhance model robustness against specific distortions, for example, adding motion blur to simulated stick objects to train the model for real motion blur scenarios. In essence, SLDA allows us to inject domain-specific distortions (such as the motion blur common to fast-moving sticks) into training examples in a controlled way, without globally distorting the image.

In addition, SLDA adjusts the inserted objects' size and resolution to match the scene's context. This is critical for low-resolution objects such as hockey sticks; an inserted stick is scaled and resampled to appear as if it were naturally part of the scene (at similar pixel size and sharpness as existing sticks). We also provide flexibility in where augmented objects are placed: SLDA's insertion function allows users to dictate specific locations and context (for example, on the ice near players' hands) for new stick instances. This ensures augmented data remains relevant and realistic, and can be tailored to model vulnerabilities (e.g., focusing on regions where sticks are often missed).

We conducted a series of experiments using a YOLOv8-seg model to evaluate SLDA under various augmentation conditions. We compared SLDA against baseline (no augmentation), traditional image-level augmentations, and other object-level methods, across different model complexities. The evaluation assessed overall model accuracy improvements and robustness to motion blur, as well as performance consistency on models of various sizes. Our results demonstrate that SLDA significantly outperforms existing augmentation methods: for instance, SLDA improved mAP@50 by 5.8% (from 86% to 92%), establishing a new benchmark in the field. Furthermore, SLDA achieved the highest F1-score across all model complexities, consistently outperforming alternative augmentation methods. Qualitative results also reveal that SLDA significantly improves the model's ability to accurately segment hockey sticks, particularly in scenarios involving motion blur.

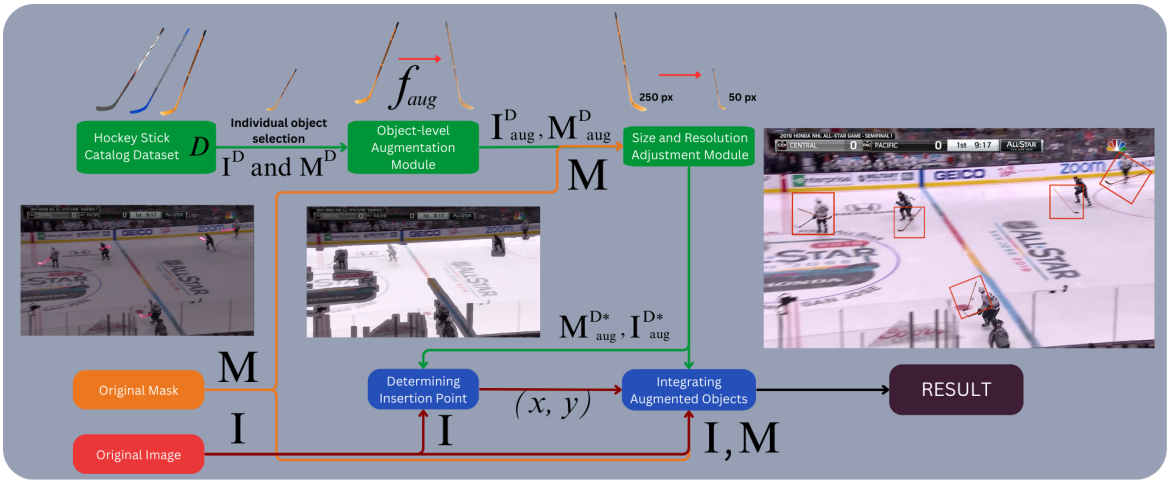


Figure 3.2: Overall workflow of the SLDA technique. A segmented hockey stick from the catalog (with its mask) is augmented with object-level transformations, then scaled and adjusted to the scene's average stick size and resolution, and finally inserted at a context-appropriate location in the scene.

3.2 Methods

The overall workflow of the Synthetic Local Data Augmentation (SLDA) technique is illustrated in Figure 3.2. SLDA takes two inputs: (1) a *Hockey Stick Catalog Dataset D* containing segmented hockey stick images, and (2) a target scene image (with its existing mask) that we aim to augment with additional sticks. The process involves three main modules applied in sequence:

1. **Object-level Augmentation Module:** Applies various transformations to a stick image from the catalog to create an augmented stick O_D^{aug}.

2. **Size and Resolution Adjustment Module:** Scales and adjusts the resolution of O_D^{aug} to match the characteristics of the target scene.

3. **Image Assembling Module:** Determines an optimal insertion point in the scene and integrates the augmented object into the scene image and mask.

3.2.1 Hockey Stick Catalog Dataset

A specialized hockey stick dataset was constructed as the source for object insertion. We gathered an initial pool of 350 high-resolution hockey stick images from a commercial online store ("Hockey Life") [1], which offers a wide range of clear product images of hockey sticks. Goalie sticks were excluded to avoid introducing objects with significantly different shape/size, which could mislead the model. Each image was manually segmented to isolate the stick from its background.

To ensure diversity without redundancy, we performed a similarity analysis on the pool and removed highly similar images, resulting in 209 unique hockey stick images. Each segmented stick image (denoted as I_D) comes with a binary mask (M_D) indicating the stick's pixels. This *Hockey Stick Catalog Dataset D* serves as the library of objects that SLDA will draw from for augmentation.

For notation clarity, let I_D and $M_D \in \{0, 1\}$ represent an image–mask pair from dataset D (with $M_D = 1$ at stick pixels). Each I_D contains exactly one hockey stick.

3.2.2 Object-level Augmentation Module

The object-level augmentation module takes an extracted stick image I_D (with mask M_D) from the catalog and produces an augmented variant $(I_D^{\text{aug}}, M_D^{\text{aug}})$. We define a suite of traditional augmentation operations (transformations) denoted $[f_1, f_2, \ldots, f_m]$, for example, rotation, flipping, Gaussian blur, and motion blur. Each transformation f_i is applied with a configurable probability p_i, forming a probability vector $\mathbf{p} = [p_1, p_2, \ldots, p_m]$. During augmentation, transformations are applied stochastically according to these probabilities, yielding a composed augmentation function f_{aug} that maps the original (I_D, M_D) to an augmented $(I_D^{\text{aug}}, M_D^{\text{aug}})$:

$$I_D^{\text{aug}},\ M_D^{\text{aug}} \ = \ f_{\text{aug}}\big(I_D,\ M_D \,\big|\, \mathbf{p}\big). \tag{3.1}$$

For this study, we are focusing on making the model more robust against motion blur, so the f_{aug} will be more biased for applying the appropriate transformation. The particular transformations and their possibilities described in Table 3.1.

Applying these transformations to the stick image and mask yields the augmented object O_D^{aug} (with image I_D^{aug} and mask M_D^{aug}). Notably, the augmentation is applied in an object-centric manner: for example, adding motion blur affects only the stick pixels,

[1] https://www.prohockeylife.com/collections/hockey-sticks

Table 3.1: Transformation parameters and probabilities.

Transformation	**Parameter**	**Prob.**
Rotate	Rotation angle: 0 to 360° (continuous)	62.5%
Flip	0 (vertical): 33.3% 1 (horizontal): 33.3% -1 (both axes): 33.3%	62.5%
Motion Blur	Blur size: 5, 7, 9, 11, (each 20%) Blur angle: 0 to 360° (continuous)	87.5%
Gaussian Blur	Kernel size: 3, 5, 7, 23 (each 25%)	62.5%

simulating a fast-moving stick while keeping the rest of the scene untouched. By controlling the type and intensity of distortions (via the probability vector P), we can generate synthetic stick instances that target specific model weaknesses (e.g., inability to handle blurred sticks).

3.2.3 Size and Resolution Adjustment Module

Before inserting segmented object $\mathbf{O^D}_{\text{aug}}$ to the image I, we need to adjust the dimensions of the inserted objects, to ensure seamless integration with the scene's existing elements and enhance the model's ability to accurately segment objects at a specific resolution.

The augmented segmented object, $\mathbf{O^D}_{\text{aug}}$, is formed by applying the mask $\mathbf{M^D}_{\text{aug}}$ to the image $\mathbf{I^D}_{\text{aug}}$ in a pixel-wise manner. This operation isolates the region of interest by zeroing out pixels outside the masked area, retaining only the details within the mask.

For size adjustments, the module calculates the average size μ_{size} of the hockey sticks already present within the scene.

This is done by averaging the lengths and widths of all detected hockey sticks $\mathbf{O}_{\text{aug},n}$ in the scene, where N is the number of objects (sticks) on the image I. The dimensions of the hockey sticks to be inserted are then scaled to this average. The scale factor is randomly chosen from a uniform distribution between -0.05 and 0.05 to account for natural size variation:

Resolution adjustments involve calculating the polygonal area of the objects currently in the scene using the Shoelace formula. This calculated area is then used to adjust the pixel density of the inserted hockey stick images.

For each existing object $\mathbf{M}_{\text{aug},n}$ represented by polygon vertices (x_i, y_i)

$$A_{\text{existing},n} = \frac{1}{2} \left| \sum_{i=1}^{k} (x_i y_{i+1} - y_i x_{i+1}) + (x_k y_1 - y_k x_1) \right| \tag{3.2}$$

The total existing area, A_{existing} is then the sum of areas of all N objects. The area of the inserted augmented object $\mathbf{O^D}_{\text{aug}}$, to be inserted, denoted as A_{inserted}, is calculated using the same polygonal area formula.

To calculate the adjustment factor for resolution, we take the square root of the ratio between the area of the inserted object A_{inserted} and the total existing area A_{existing}.

The resolution of the inserted object is modified by multiplying its original resolution with the adjustment factor.

3.2.4 Image Assembling Module

The Image Assembling Module encompasses two final steps in the augmentation process: determining the insertion point and integrating augmented objects into the scene.

The first stage focuses on calculating the optimal spatial coordinates where the augmented object $\mathbf{O}^{\mathbf{D}}_{\text{aug}}$ will be placed within the scene. The coordinates are determined based on the characteristics of the scene and the specific properties of the augmented object. We denote the function responsible for this as $f_{insert}(I, \mathbf{O}^{\mathbf{D}}_{\text{aug}}) \rightarrow (x, y)$.

For this study, which evaluates the efficiency of our data augmentation on the YOLOv8 model—a model that primarily processes visual information without considering non-visual object relationships—the focus of the insertion points is on ensuring high visibility. The preferred location for hockey sticks, therefore, is generally on the ice field, where they are most easily seen. Therefore we indentify, f_{insert} as follows:

1. **Identify White Areas:** Analyze the current scene I to identify potential ice areas, typically white or very light in color, using color thresholding techniques.
2. **Use Dilation to Merge Segmented Field Parts:** Merge the segmented parts of the ice field using image dilation techniques to reduce fragmentation and create a contiguous potential insertion area.

3. **Filter Out Small Artifacts:** Select the largest connected component within the dilated area to eliminate the possibility of selecting irrelevant white parts, such as those from advertising or other artifacts.

4. **Apply Erosion to Ensure a Buffer from Edges:** Create a buffer zone around the edges of the identified ice field using erosion to prevent the inserted object from overlapping with the scene's boundaries.

5. **Random Coordinate Selection within the Selected Area:** Select random coordinates (x, y) within the prepared area.

Once the augmented object image $\mathbf{I}^{\mathbf{D}}_{\text{aug}}$ and its corresponding mask $\mathbf{M}^{\mathbf{D}}_{\text{aug}}$ are prepared and the insertion coordinates (x, y) determined, the object is integrated into the scene image I and the scene mask M.

Average time for data augmentation is approximately 0.24 seconds per image, with an average of 4 sticks inserted per image. The full insertion process, including transformations, takes about 0.04 seconds per stick, with transformations alone requiring 0.03 seconds. The baseline insertion time (without transformations) is 0.01 seconds. In our case, augmenting a 1000-image dataset took 4 minutes.

3.3 Experiments

3.3.1 Experiment Setup

To evaluate SLDA, we curated a custom hockey stick segmentation dataset of game footage. It includes frames from a 2019 NHL playoff game (Eastern Conference Finals, BOS vs. CAR) and the 2019 NHL All-Star Game. In total, we collected 1000 HD images (1280×720) with exhaustive manual annotations of hockey sticks, yielding 4226 segmented stick instances. We randomly split this dataset into 70% training, 20% validation, and 10% testing sets. Notably, since no public hockey stick segmentation dataset existed, we plan to release this dataset alongside our work to support the community.

Augmentation strategies were selected to benchmark the effectiveness of SLDA against both traditional and other object-level augmentation methods:

No Data Augmentation (No DA): Baseline model trained on the raw training set with no augmentations. This provides a reference performance to assess gains from augmentation.

Traditional Augmentation (TA): Classic image-level augmentations applied (see Table 3.2), such as rotation, grayscale conversion, hue shifts, blur, and noise, one per augmented image. This tests if standard whole-image perturbations help the model.

Copy-Paste Augmentation: An object-level augmentation where segmented stick instances (from our catalog) are randomly pasted into training images (without SLDA's additional transformations). Inserted objects were placed in random positions within a central square of the image (excluding a 110-pixel border) and scaled to $\sim$ 100 pixels in length. This represents the existing object-level approach from Ghiasi et al. [24] in our context.

SLDA (Synthetic Local Data Augmentation): Our proposed method.

Combined Augmentations: Two combinations were tested – Copy-Paste + Traditional and SLDA + Traditional – to see if mixing object-level and image-level augmentations yields further benefits.

The ObjectAug method was not included in our experiments, as no official or reliable implementation is currently available online.

Traditional data augmentation techniques are described below in Table 3.2, which details the specific transformations along with their parameters:

Table 3.2: Traditional Data Augmentation Parameters

Transformation	**Parameter**
Rotation	Angle range: -15° to 15°
Grayscale	Intensity: Up to 14%
Hue Adjustment	Range: -48° to 48°
Blur	Maximum blur: 1.5px
Random Noise	Up to 1.5% of pixels

This approach expanded the training dataset to three times its original size by applying one transformation per augmented image.

For the Copy-Paste method, objects were inserted into a square area that covered the entire image but excluded a 110-pixel margin from each border. The size of the inserted objects was adjusted to 100 pixels.

The YOLOv8s-seg model, a variant of YOLOv8 optimised for segmentation tasks, was used to evaluate the effectiveness of the augmentation strategies. Among the YOLO family variants we tested (v7, v8, and v9), YOLOv8 consistently achieved the best segmentation

accuracy on our hockey stick dataset. We employed the small variant (YOLOv8-seg) to balance accuracy and computational efficiency. While other segmentation architectures (e.g., Mask R-CNN or transformer-based models such as SAM) exist, YOLO models provided a stronger trade-off between training efficiency and segmentation performance on our relatively small dataset, making YOLOv8s-seg the most practical choice for this study.

Training details for the model include a training duration of 50 epochs with a batch size of 8. All other parameters were set to their default values. The augmentation techniques were only applied to the training data, with the validation and test sets left unchanged to maintain evaluation consistency.

3.3.2 Quantitative Results

We evaluated each model on the test set using standard detection/segmentation metrics. The key metrics reported are mean Average Precision at 50% IoU (mAP@50) and at 50–95% IoU (mAP@50–95), along with Precision, Recall, and F1-score (for the stick class). Table results reflect mask-based segmentation performance (i.e., evaluating the segmentation masks of sticks).

1) mAP@50 Results

Table 3.3 shows the mAP@50 (mask) achieved by models under each augmentation condition:

Table 3.3: Comparison of mAP50 (Masks) for Different Augmentation Conditions

Configuration	**mAP50 (Masks)**
No DA	0.842
Traditional DA	0.850
Copy-Paste DA	0.885
SLDA	**0.895**
Copy-Paste + Traditional	0.926
SLDA + Traditional	**0.933**

SLDA alone achieved the highest mAP@50 of all single-method conditions. The best result overall was attained by combining SLDA with traditional augmentation, slightly higher than combining copy-paste with traditional. This suggests that SLDA's benefits are complementary to image-level techniques; using both yields an additive improvement.

2) mAP@50–95 Results

Table 3.4: Comparison of mAP@50–95 (Masks) for Different Augmentation Conditions.

Configuration	mAP@50–95 (Masks)
No DA (Baseline)	0.413
Traditional DA	0.418
Copy-Paste DA	0.442
SLDA	**0.453**
Copy-Paste + Traditional	0.493
SLDA + Traditional	**0.504**

Table 3.4 displays the mAP50-95 values for each configuration. The results indicate that SLDA achieved higher mAP50-95 compared to the baseline and other standalone augmentations. When combined with Traditional Data Augmentation, SLDA achieved the highest mAP50-95, surpassing all other configurations.

3) Precision, Recall, and F1-Score

To further analyze performance, Table 3.5 provides the Precision, Recall, and F1-score of the YOLOv8s-seg model under each augmentation scenario (averaged over the test set):

Table 3.5: Performance metrics for YOLOv8s under different data augmentation scenarios.

Configuration	Precision	Recall	F1-Score
No DA (Baseline)	0.895	0.784	0.837
Traditional DA	0.913	0.786	0.845
Copy-Paste DA	0.904	0.813	0.856
SLDA	**0.924**	**0.834**	**0.877**
Copy-Paste + Traditional	0.945	0.864	0.903
SLDA + Traditional	**0.969**	**0.880**	**0.922**

SLDA shows the highest Precision (0.924), Recall (0.834), and F1 (0.877) among single-method augmentations. Compared to the baseline, SLDA improved Recall by ∼5% (0.834 vs 0.784) and Precision by ∼3% (0.924 vs 0.895), resulting in an F1 increase from 0.837 to 0.877. SLDA also outperformed Copy-Paste augmentation (F1 0.877 vs 0.856). These gains

indicate that SLDA-trained models are better at detecting sticks (higher Recall) without sacrificing precision (false positives remain low, Precision increased). Among all conditions, the combination of SLDA + Traditional yielded the best scores (Precision 0.969, Recall 0.880, F1 0.922), suggesting that SLDA's object-level focus and traditional augmentations' image-level diversity address complementary aspects of the problem. In summary, SLDA provides the strongest performance across all three metrics, demonstrating its effectiveness in improving both the accuracy of stick identification and the model's ability to find more sticks overall.

4) Validation Segmentation Loss Comparison

Beyond final accuracy, we examined training dynamics by plotting the validation segmentation loss over epochs for each augmentation scenario (Figure 3.3). This analysis reveals how each augmentation influences the model's learning stability and convergence:

The No Augmentation (Baseline) model showed the most erratic training. Its validation loss fluctuated heavily and settled at a higher minimum loss than any augmented counterpart. This instability likely reflects the model struggling with limited and repetitive training examples (e.g., overfitting certain stick appearances and failing on others).

Traditional Augmentation initially lowered the loss (by providing more varied images), but over training epochs it did not yield significant further improvements. Its loss curve still showed notable oscillations, indicating that image-level transforms alone might not address the core difficulties (such as blurred or tiny sticks in context).

Copy-Paste Augmentation achieved a better minimum loss than Traditional (the curve goes lower), suggesting that introducing new stick instances helps the model learn more general representations. However, the Copy-Paste curve had more pronounced fluctuations than SLDA's curve, implying less stable training. The lack of size/resolution adjustment in naive copy-paste may introduce inconsistencies that the model grapples with (hence bumpier convergence).

SLDA yielded the lowest validation loss and the smoothest curve among single methods. This indicates not only better performance but a more stable learning process. By providing realistic, context-matched augmented examples, SLDA likely reduced confusion for the model (e.g., it sees blurred sticks in plausible settings, improving robustness and confidence, which translates to a steady loss decline). SLDA's curve suggests it helped avoid the severe overfitting/underfitting swings seen in baseline training.

In summary, SLDA led to faster and more stable convergence, and the final model had the lowest segmentation error on validation data. Copy-Paste and Traditional aug-

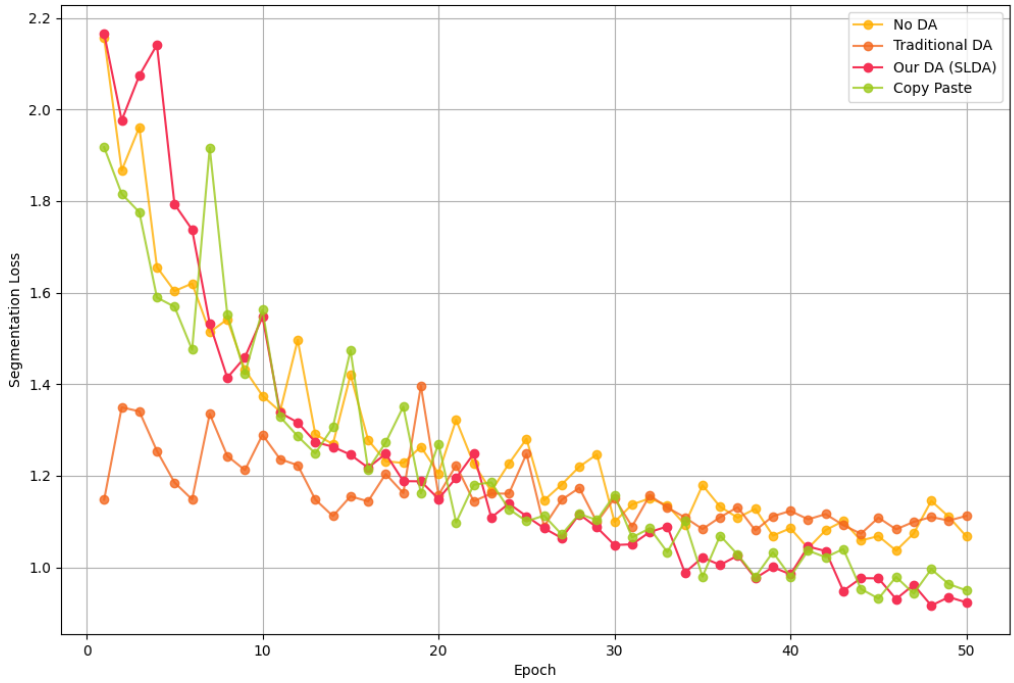


Figure 3.3: Validation Segmentation Loss over Training Epochs, Comparing Different Data Augmentation Scenarios.

mentations provided some benefit over none, but were less effective than SLDA and even combined did not match SLDA's stability.

5) Comparative Effectiveness of SLDA Across Model Complexities

We focuses on the performance impact of Synthetic Local Data Augmentation (SLDA) across various complexities of the YOLOv8 Segmentation model, highlighting the consistent efficiency of SLDA-enhanced models across all sizes when compared to Copy-Paste, a comparative object-focused DA module.

The tables 3.7, 3.6 illustrate that models augmented with SLDA not only perform better in their respective size category but also often outperform larger models augmented with Copy-Paste technique.

Table 3.6: Comparison of mAP50 (Masks) Across Architecture Sizes.

Model Size (YOLOv8)	**DA**	**mAP50 (M)**
Large (L)	SLDA	**0.920**
	Copy-Paste	0.862
Medium (M)	SLDA	**0.913**
	Copy-Paste	0.856
Small (S)	SLDA	**0.895**
	Copy-Paste	0.885
Nano (N)	SLDA	**0.834**
	Copy-Paste	0.79

3.3.3 Qualitative Results

We provide a qualitative comparison of model outputs to illustrate how SLDA augmentation improves segmentation in challenging scenarios.

These examples focus on difficult conditions such as motion blur, partial occlusion, and low lighting, highlighting differences in detection capability.

In general, the SLDA-augmented model produces more complete and accurate stick masks under various conditions, whereas other models often miss or partially segment the sticks. The improvements are especially pronounced for blurry sticks. For instance, in scenes where a hockey stick moves rapidly causing blur, the SLDA model is often able to detect and segment it, whereas the baseline and traditionally augmented models completely miss it, and the Copy-Paste model might catch only a fragment.

To demonstrate, Figure 3.4 shows side-by-side outputs of the different models on a representative frame with a clear central action. All models are tasked with segmenting the hockey sticks present.

In the example of Figure 3.4, even though none of the models can segment an extremely motion-blurred stick in the distance (an extremely difficult case), the SLDA-augmented model shows the best overall performance. It successfully segments sticks that are moderately blurred, maintaining more continuous and complete masks. In contrast, the Copy-Paste model, despite being the second-best quantitatively, performs noticeably worse in this qualitative comparison. The Copy-Paste model's masks are less uniform and it sometimes only identifies part of the stick (e.g., just the blade) while missing the shaft. The baseline

Table 3.7: Precision, Recall, and F1-Score Across Architecture Sizes. Where CP stands for Copy-Paste.

Model Size	DA	Precision	Recall	F1-Score
Large(L)	SLDA	**0.952**	**0.859**	**0.903**
	CP	0.926	0.789	0.852
Medium(M)	SLDA	**0.947**	**0.856**	**0.899**
	CP	0.915	0.789	0.847
Small(S)	SLDA	**0.924**	**0.834**	**0.877**
	CP	0.904	0.813	0.856
Nano(N)	SLDA	0.876	**0.761**	**0.815**
	CP	**0.938**	0.718	0.813

and traditional models (not shown in the figure here) fare even worse, often missing the sticks entirely or producing very fragmentary masks in such challenging scenes.

Overall, the qualitative evidence aligns with our quantitative findings: SLDA leads to more robust and precise stick segmentation, particularly in cases that mimic our augmentation focus (motion blur, varied orientations, etc.). The augmented model has essentially learned to "see" sticks that would otherwise be invisible or indistinct, thanks to the realistic synthetic examples it was trained on.

3.3.4 Visual Comparison of Different Augmentation Scenarios

To further illustrate the effect of each augmentation strategy, we compiled a series of visual comparisons (beyond Fig. 3.4) showing typical outputs:

No Augmentation (Baseline): Frequently misses sticks that are not very clear or are partially obscured. Its detections are mostly limited to obvious, fully visible sticks. Blurry or small sticks are rarely detected.

Traditional Augmentation: Shows marginal improvement over baseline in some cases (perhaps catching a small stick here or there), but still fails on motion-blurred sticks. Since it never saw isolated blur on sticks during training (blur was applied globally), it doesn't reliably handle that scenario.

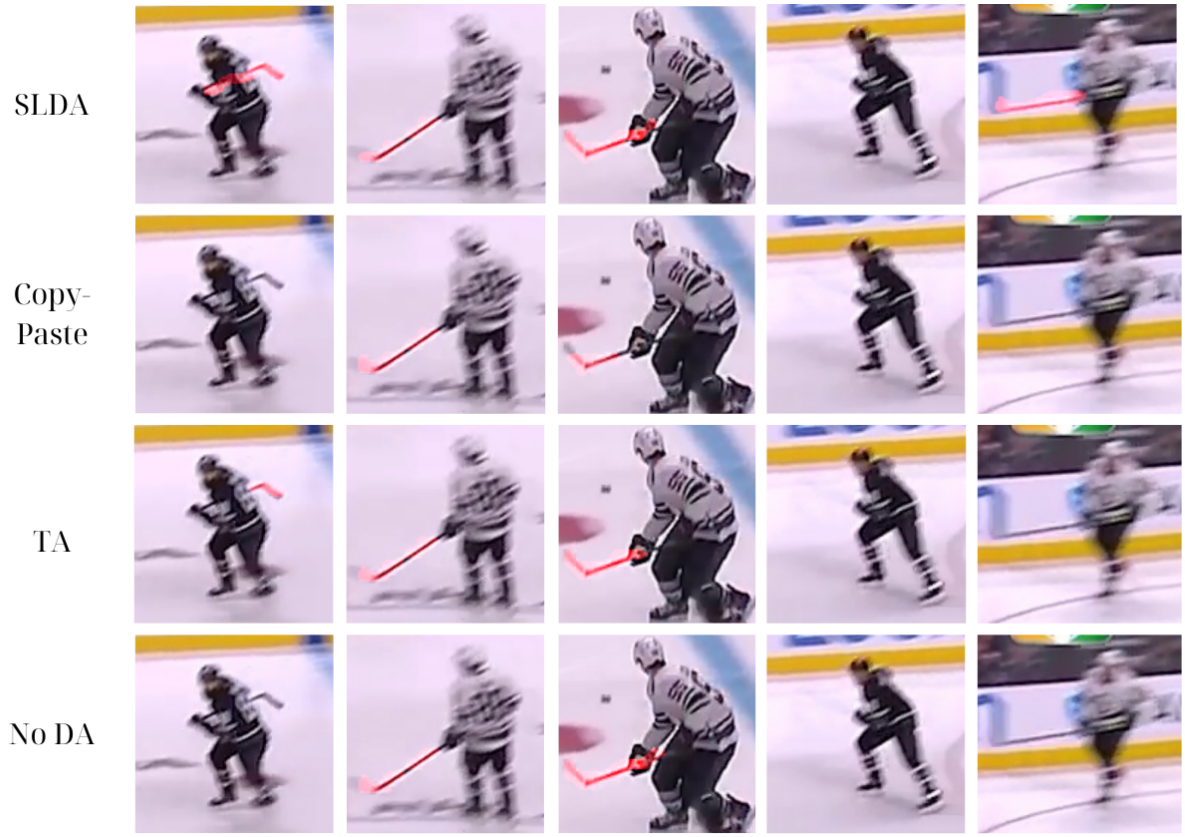


Figure 3.4: Comparison of hockey stick segmentation by different models on the same scene. (Only the clear scene is shown here for brevity; each model's output is outlined.) From top to bottom: SLDA Augmentation, Copy-Paste Augmentation, Traditional Augmentation, No Augmentation (Baseline).

Copy-Paste Augmentation: Detects more sticks than baseline, including some partially occluded ones due to having seen more stick instances. However, if the inserted sticks during training didn't mimic blur or lighting changes, the model still struggles with those. Qualitatively, its segmentations can appear incomplete, e.g., identifying one end of a stick but not the whole thing, suggesting the model is unsure when the appearance deviates (such as blurred midsections).

SLDA Augmentation: Consistently detects sticks in a wider range of conditions. When a stick is motion-blurred, the SLDA model often still marks an approximate region, whereas

other models leave it blank. For sticks near players or the boards, the SLDA model is better at separating the stick from the player's body or background, likely because it saw many examples of sticks in diverse contexts (due to insertion at various locations).

These visual comparisons reinforce that SLDA's context-aware, object-level augmentation produces a model with a more holistic understanding of what a hockey stick can look like. It captures the shape even when distorted by motion or partially hidden, whereas models without such targeted augmentation see those cases as out-of-distribution.

3.4 Conclusion

In this chapter, we presented Synthetic Local Data Augmentation, a novel data augmentation technique for semantic segmentation of hockey sticks at the object level. SLDA combines the strengths of the two prior object-level augmentation paradigms, Copy-Paste and ObjectAug, by both introducing new object instances and applying custom local transformations to them. This hybrid approach addresses the limitations of each: Copy-Paste's lack of scale adjustment and ObjectAug's lack of new object variety.

Through comprehensive experiments with YOLOv8-seg, we evaluated SLDA under various augmentation conditions. The results demonstrated clear improvements in model performance (higher precision, recall, F1, and mAP) and, importantly, enhanced robustness to motion blur – a major challenge in hockey analytics. SLDA not only improved overall accuracy but specifically helped the model handle blurry, small, or occluded sticks better than other augmentation methods. It achieved the best results across all model sizes and metrics in our case study.

Additionally, we showed that SLDA outperforms existing object-level augmentations (such as Copy-Paste) across all model complexities. Even the smallest model benefited greatly from SLDA, closing much of the gap to larger models. This suggests SLDA can be a valuable tool for training efficient models without sacrificing accuracy.

In summary, SLDA offers a simple yet effective way to inject domain-specific variations (such as motion blur for fast objects) into training data. By doing so, it significantly boosts a model's ability to generalize to challenging real-world conditions, as evidenced by our hockey stick segmentation results. The technique is not limited to hockey sticks; it can be applied to other tasks where one has a library of segmented objects and needs to simulate contextual distortions, making it a broadly applicable strategy in the arsenal of data augmentation techniques for computer vision.

Looking ahead, while SLDA strengthens object-level representations, many errors in hockey vision arise from missing scene context such as player–puck interactions and camera geometry. The next chapter, Chapter 4, addresses this limitation by introducing a Unity-based simulation tool that reconstructs plays in 3D and generates controllable multi-view video, complementing SLDA's local augmentation with scene-level synthesis.

Chapter 4

Scene-Generating Simulation Tool (Unity Project)



Figure 4.1: **(a)** Frame from a broadcast video used as input for scene reconstruction.



Figure 4.2: **(b)** Corresponding frame from the Unity-based simulation output.

Following the object-level gains achieved in Chapter 3, this chapter presents a scene-generating simulation tool that reconstructs hockey plays in Unity from structured data or broadcast video. The tool complements SLDA by addressing errors that arise from limited scene context, including camera geometry, multi-view variation, and the team- and puck-level formation on the rink, while also producing controllable, labeled video at scale. We describe the system design covering input handling, homography alignment, tracking, and rendering; demonstrate dual input modes (tabular and video); and detail synchronized annotation outputs for downstream training and evaluation.

4.1 Tool Purpose and Functional Overview

The goal of this tool is to bridge the gap between symbolic representations of gameplay, such as player and puck trajectories, and visually grounded simulation. It enables users to generate fully rendered hockey scenes in Unity based on structured tabular data or processed broadcast video, supporting downstream applications such as visual model training, scenario playback, and simulation-based data augmentation.

This tool is intended to connect symbolic gameplay data, such as player and puck positions, with visual simulations rendered in Unity. It allows users to generate complete hockey scenes based on structured inputs or processed broadcast footage. The resulting output can be used for various downstream tasks, including model training, scenario replay, or augmenting datasets.

At a high level, the system converts player and puck positions into temporally synchronized 3D renderings on a standardized rink environment. It supports both *synthetic simulation* (from manually constructed or algorithmically generated trajectories) and *scene recreation* (from actual broadcast videos), offering a flexible interface for generating realistic hockey sequences at scale.

Key features include:

- *Dual-Mode Input Compatibility:* Users can input data either through structured tables (e.g., for simulated or symbolic play) or via broadcast video (for visual extraction and recreation of real games). When using video-only input, the system includes a built-in annotation tool to manually label puck positions frame-by-frame, with interpolation support for skipped frames.

- *Homography Alignment:* All player and puck coordinates are projected into a unified coordinate system defined by a fixed rink template included with the tool.

- *Automatic Ground Truth Generation:* Each rendered simulation produces synchronized annotations for player bounding boxes (in both broadcast and rink coordinates), per-frame player positions, and full-resolution player segmentation masks. This enables direct downstream use for visual model training, evaluation, or benchmarking.

- *Configurable Visual Appearance:* The rendering engine supports team color configuration, and dynamic camera manipulation to match a variety of playback or training conditions.

- *Modular Architecture:* The system is structured into independent modules for ingestion, homography alignment, tracking, and rendering—allowing for reuse, substitution, or future expansion (e.g., automated puck detection).
- *Python–Unity Integration:* All data pre-processing and transformation logic is implemented in Python, while the simulation and rendering logic runs in Unity (C#). A unified command-line interface (CLI) links the full pipeline.

While player positions can be estimated automatically, puck trajectories must currently be either provided in tabular format or annotated manually via the included labeling tool. A future version of the system may incorporate automated puck estimation based on contextual cues or deep learning-based methods.

The tool is designed to support two primary research workflows: **(1)** *Scene-Generating Data Augmentation* - synthetic gameplay scenes can be generated from symbolic or constructed data, enabling enriched training sets for visual models. **(2)** *Scene-Preserving Simulation* - real-world gameplay can be reconstructed from video to enable analytics, visual debugging, or model benchmarking in realistic conditions.

4.2 System Architecture

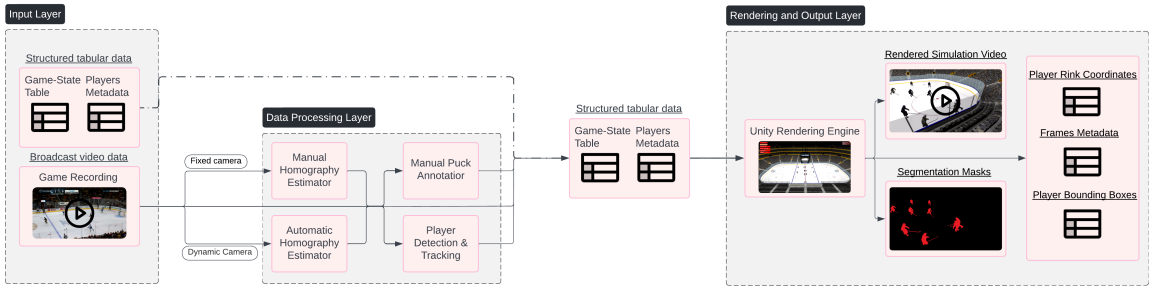


Figure 4.3: Overview of the system architecture. The structured tabular data block may either be directly provided as input or derived from broadcast video data after processing.

The tool follows a modular and loosely coupled architecture, organized into three core layers: input handling, data processing, and rendering/export. These layers are coordinated via a Python-based orchestration layer that manages input parsing, homography transformation, player tracking, and inter-process communication with Unity through a command-line interface (CLI).

4.2.1 Input Handling Layer

The system supports two distinct input modes, corresponding to different data availability scenarios: (1) structured tabular data and (2) broadcast video. Both paths are ultimately converted into a unified internal representation—referred to as the *game-state table*—which encodes all necessary information to drive rendering and animation.

Structured Tabular Input

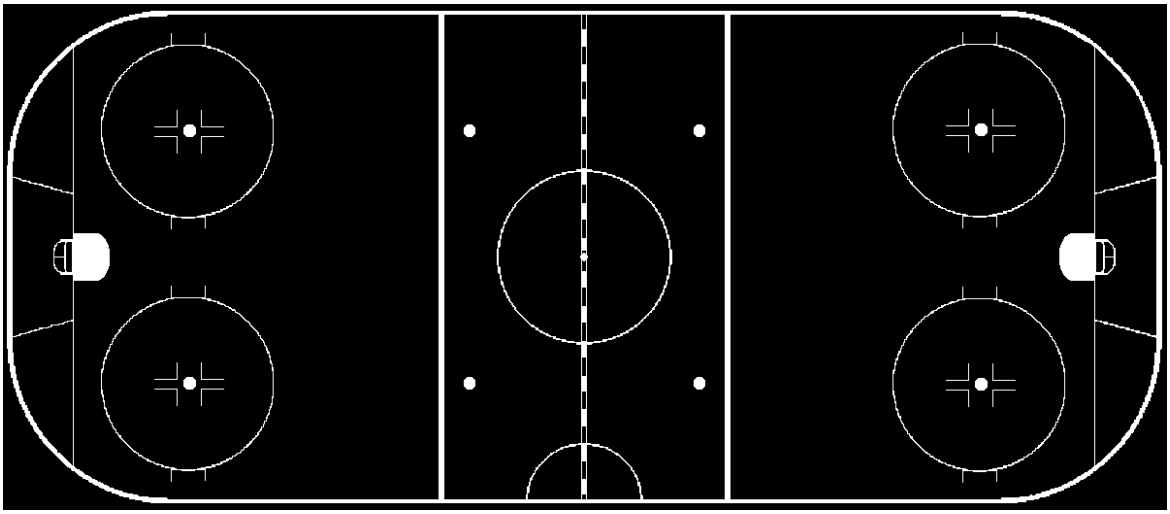


Figure 4.4: Rink template used as the coordinate system for all player and puck annotations. Positions in structured input tables are defined relative to this space.

The primary input format consists of a pair of structured tables that describe the positions and identities of players and the puck over time. These inputs are expressed in a coordinate system defined by a custom rink template included with the tool, which serves as a spatial reference standard throughout this thesis (see Figure 4.4).

Frame-by-Frame Game-State Table This table encodes gameplay over time, with one row per timestamp. Each row includes 2D X-Y coordinates for every player and for the puck, structured either by fixed column layout (e.g., `P1_X`, `P1_Y`, ..., `Puck_X`, `Puck_Y`) or by indexed player IDs. A simplified excerpt is shown in Table 4.1.

Table 4.1: Input: Example of Game-State Table

Timestamp	P1_X	P1_Y	P2_X	P2_Y	...	Puck_X	Puck_Y
00:00:01	23.1	10.7	35.2	14.3	...	42.5	16.8
00:00:02	24.2	11.1	36.0	15.0	...	43.7	17.1

Player Metadata Table This secondary table maps internal player identifiers (e.g., P1, P2, ...) to jersey numbers and team affiliations. This enables realistic rendering with visually consistent team colors and uniform designs, as shown in Table 4.2.

Table 4.2: Input: Example of Player Metadata Table

Player Index	Jersey Number	Team
P1	91	Home Team
P2	7	Away Team

Broadcast Video Input

When tabular tracking data is unavailable, users may supply a *broadcast video* of the game instead. The system supports two types of video inputs, each requiring different preprocessing steps based on camera stability.

Fixed-Camera Broadcast For footage captured from a stationary camera, the user must first define a *manual homography transformation* that maps image-space (pixel) coordinates to the rink template coordinate system. A custom Python tool enables the user to interactively define point correspondences between key rink features (e.g., faceoff circles, blue lines) and their real-world locations. This transformation ensures that any player detected in the image can be spatially aligned with the internal rink coordinate system.

Once the homography is defined, the system applies the following pipeline:

1. **Player Detection and Tracking:** Uses an external vision model (wrapped in Python) to detect player bounding boxes and track their movements across frames.
2. **Homography Projection:** Maps each tracked player's position from image space into the rink template coordinate system.

3. **Manual Puck Annotation (Optional):** When puck trajectories are unavailable, users can manually annotate puck positions using a built-in frame-labeling tool. The tool allows frame-by-frame selection with interpolation for skipped frames, enabling fast annotation with minimal overhead.

The outputs of these steps are compiled into the same two-table format used in the structured tabular input mode.

Moving-Camera Broadcast For dynamic broadcast videos with a moving or panning camera, the user can bypass manual homography estimation. The system includes an *automated homography estimator* that generates a per-frame transformation matrix by analyzing rink features or visual anchors. The remainder of the pipeline, player tracking, coordinate projection, and optional manual puck annotation, remains identical to the fixed-camera case, producing a standardized game-state table suitable for rendering.

4.2.2 Data Processing Layer

The data processing layer transforms parsed input, whether structured tables or video-derived coordinates, into a temporally synchronized, spatially aligned game-state representation suitable for 3D rendering. This layer operates on all positional data, ensuring that each timestamped entity (player or puck) is accurately projected into a canonical rink coordinate system.

The processing pipeline consists of three key modules:

- **Homography Estimation and Projection**
- **Player Detection and Tracking**
- **Manual Puck Annotation (Optional)**

Homography Estimation and Projection. For video-based input, homography projection is the first and most critical step in aligning detections with the canonical rink coordinate system. The system supports two modes of operation based on camera behavior:

- **Manual Homography Tool (Static Camera):** For footage captured with a fixed-angle camera, the system includes an interactive GUI that allows users to define point correspondences between video frames and a standard rink template. This transformation is assumed to be static across the clip and is applied uniformly to all frames.

- **Automatic Homography Estimator (Dynamic Camera):** For dynamic or broadcast-style footage with panning or zooming, the system employs an automated homography estimation module based on the method proposed by Mehrnaz Fani in *Localization of Ice-Rink for Broadcast Hockey Videos*[37].

  In this approach, a ResNet18-based regressor is trained to localize four control points on the rink model for each video frame. These predicted control points are smoothed using a Hann-window-based convolution over time to reduce jittering across successive frames. The final homography matrices are then computed using the Direct Linear Transform algorithm based on the smoothed control point pairs. This pipeline enables robust and smooth localization of the broadcast view relative to the rink template, even under significant viewpoint shifts or shot transitions.

Once homography matrices are available (either manually or automatically), player coordinates and any manually labeled puck positions are projected into rink space to produce spatially consistent data across modalities.

Player Detection and Tracking. This module builds upon the multi-object tracking architecture described in *Broadcast Is All You Need* by Harish Prakash[40]. Player detection is first performed frame-by-frame using a standard vision model, after which detections are projected into the canonical coordinate system using the previously computed homographies.

Tracking is formulated as a bipartite graph-matching problem, where identity assignment across frames is resolved via message passing on a spatial graph. This approach is well-suited to the occlusion-heavy, fast-paced nature of hockey gameplay. The module outputs a sequence of player positions in rink coordinates, which are then populated into the game-state table.

Manual Puck Annotation (Optional). Since automated puck detection is not currently integrated, puck trajectories must either be provided in the input tables or manually labeled. To support the latter case, the system provides a Python-based annotation tool that allows users to select puck positions frame-by-frame in video input.

The annotated puck trajectory is exported in a table format and merged with the output of the tracking module to form the final game-state table.

Input.

- **Structured Pathway.** When structured tracking data is provided directly as tables (e.g., from existing analytics pipelines or simulation tools), the data processing layer performs schema validation and timestamp alignment. Since all coordinates are already expressed in rink space, no projection is required. The resulting data is reformatted into the standardized internal structure and passed to the rendering layer. In this mode, player metadata is typically supplied alongside the tracking data.
- **Player Metadata (Video Pathway).** Unlike structured input mode, where metadata is typically included, broadcast video pipelines do not inherently provide player identity or team information. Therefore, users must manually define a simple metadata table mapping each tracked player index (e.g., P1, P2, ...) to their team affiliation. This ensures accurate rendering of uniforms and team-based color coding in the Unity simulation.

Output. The data processing layer produces two synchronized outputs:

- **Game-State Table:** A timestamped record of all player and puck positions in rink coordinates.
- **Player Metadata Table:** Mapping of internal player IDs to teams, used for visual styling during rendering.

These outputs drive the animation and rendering pipeline implemented in Unity.

4.2.3 Rendering and Output Layer

The rendering engine visualizes the processed game-state data using the Unity platform. It instantiates a 3D hockey scene composed of prebuilt rink and player assets [15, 56] (see Figure 4.10, 4.11), and animates each entity based on spatial and temporal cues derived from the input.

Player orientation and motion are inferred heuristically from positional data, enabling direction-aware skating and puck-tracking behavior. Inferred actions—such as shooting—are triggered based on rule-based conditions involving puck velocity and player proximity.

Camera behavior and export settings are configurable, supporting both fixed broadcast-style views and dynamic tracking. Final output can be rendered to high-resolution video or frame sequences for analysis or presentation.

In addition to video output, the rendering engine generates synchronized auxiliary annotations that are automatically exported alongside each rendered clip. These include:

- (1) per-frame 2D bounding boxes of all players in both video-space and rink-space coordinates
- (2) player positional metadata (X, Y) aligned with the rink template
- (3) high-resolution player segmentation masks.

These annotations are saved in structured CSV and PNG formats, enabling immediate downstream use in visual training pipelines, model benchmarking, or video-based analytics.

Detailed animation logic, asset configuration, and interaction heuristics are described in 4.4 sections.

4.3 Detailed Tool Descriptions

4.3.1 Manual Homography Estimation Tool

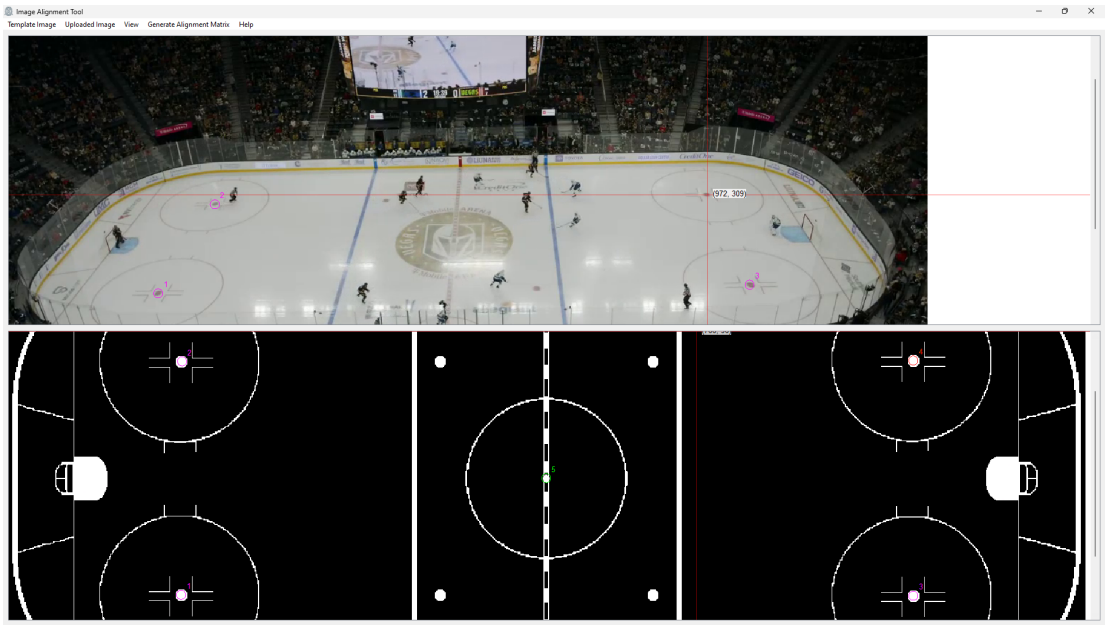


Figure 4.5: Matching points between uploaded image and template. Unassigned points are green; the next point in sequence is red.

The **Manual Homography Estimation Tool**, internally named the `Hockey Field Mapper`, is a standalone Python-based GUI application designed for frame-to-template alignment in cases where automated homography estimation is not reliable. It is typically used when the video is recorded from a fixed camera with a known or partially known perspective.

This tool enables the user to assign control points between a standardized top-down rink template and a broadcast frame. Once enough correspondences are established, a homography matrix is computed and exported for use in downstream tracking and rendering steps.

```
python hockey_sim_manual_homography.py --input_dir <path>
                                       --output_json <file> [options]
```

Template Preparation and Matching Workflow

The tool loads a standardized rink template and allows the user to define key landmark points (e.g., faceoff circles, crease corners, blue lines). Users have two options for assigning these control points:

- **Use predefined point sets** for common views (top-down, side-angle), which simplifies the matching process by following a known point order.
- **Define a custom point set** if the camera angle is unique or non-standard. Users can iteratively refine their own points to improve alignment.

This flexibility allows the tool to support a wide range of input conditions. Predefined sets provide speed and convenience, while manual configurations allow fine-grained tuning when needed.

Interactive Point Assignment The system supports standard interactive controls:

- `Right-click` to add a point on the template or uploaded image
- `Alt + Right-click` to delete a point
- `Left-click + drag` to reposition
- `Mouse wheel` to zoom, and drag background to pan
- `Right-click on a point` to rotate its index label

Unassigned template points appear in green; the next point to be assigned is highlighted in red. Once both template and image points are matched, they turn magenta, allowing users to visually confirm correct pairing.

Homography Matrix Generation Once all required correspondences are defined, the user selects `Generate Alignment Matrix (Ctrl+G)` from the menu. The computed matrix is stored internally and can be exported as a `.npy` file for reuse.

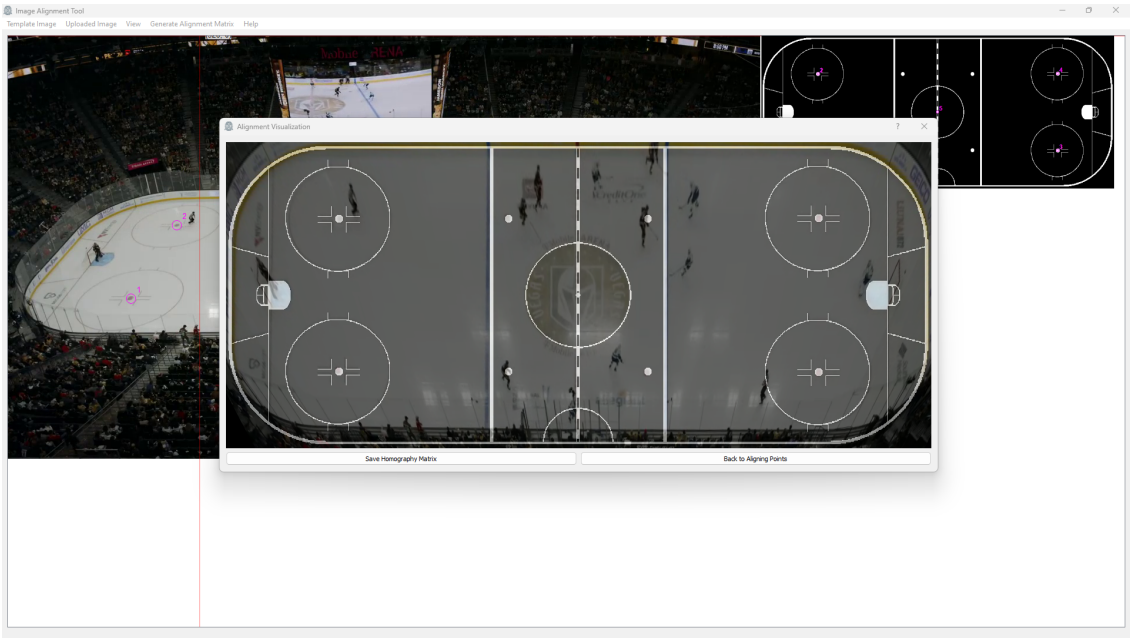


Figure 4.6: Preview of alignment result using visual overlay. This assists the user in validating the homography before export.

This visualization allows the user to confirm whether the key rink elements (e.g., lines, circles) from the template properly align with the warped input image. If alignment is poor, points can be re-adjusted before export.

Additional Features

- **Predefined and custom point sets**: Choose from known configurations or build your own for fine control.
- **Point export/import**: Save or load point sets via `Export Points (Ctrl+E)` and `Import Points (Ctrl+I)`.
- **Split view modes**: Options include 50/50 view, fullscreen template view, or fullscreen input image view with embedded template overlay.
- **Re-upload image**: Replace the input frame without clearing existing annotations.

4.3.2 Manual Puck Annotation Tool

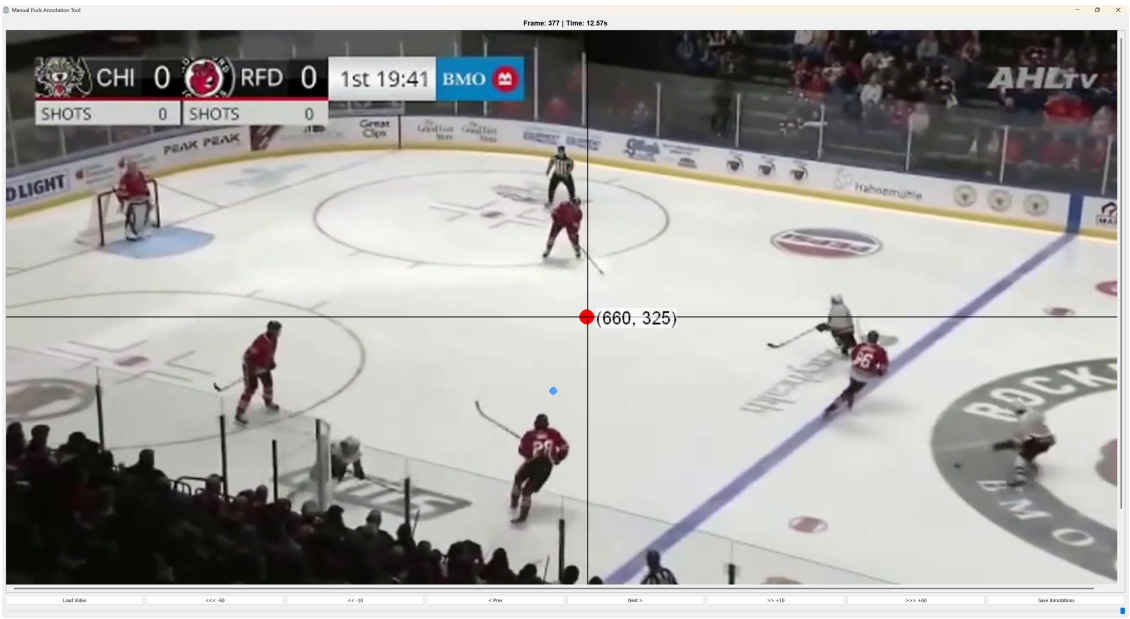


Figure 4.7: Manual puck annotation Interface. Users click to mark puck positions for each frame. Additional cues and navigation features enhance accuracy and efficiency.

The **Manual Puck Annotation Tool** is a Python-based graphical interface for manually marking puck locations in raw hockey video footage. This tool plays a vital role in the video-based scene reconstruction pipeline, enabling the creation of accurate puck trajectories in scenarios where no positional metadata is available. It is designed for usability and precision, making it possible to annotate hundreds or thousands of frames with visual feedback, fast navigation, and error correction tools.

The tool can be invoked via the following command:

```
python hockey_sim_puck_annotate.py --input_dir <path>
                                   --output_json <file> [options]
```

Purpose and Context

In our system, this tool provides the only means of recovering puck positions in broadcast or fan-recorded videos. These annotations are used downstream to drive animation logic (e.g., puck movement, shot inference) and to enrich the realism of simulated 3D renderings.

While player positions may be inferred from visual trackers, puck location, due to its small size, high speed, and frequent occlusions, must be annotated manually. This tool provides the necessary precision and speed for that task.

Key Features

- **Frame Navigation:**
  - Navigate one frame at a time with `< Prev` and `Next >`.
  - Jump by `±10` or `±60` frames using `<< -10`, `<<< -60`, `>> +10`, `>>> +60`.
  - Current frame number and timestamp (in seconds) displayed at top of interface.
- **Annotation Input:**
  - `Left-click` on the puck to annotate its position.
  - `Ctrl + Left-click` or re-clicking near an existing annotation deletes it.
  - Annotated positions appear as red dots on the frame.
- **Annotation Feedback:**
  - Blue dots show puck positions from the previous 10 annotated frames.
  - Green dots show puck positions from the next 10 annotated frames.
  - Visual cues support smooth trajectory annotation and outlier detection.
- **Precision Tools:**
  - Pixel-level **crosshair and coordinate display** under mouse pointer.
  - **Zoom slider** (10%–300%) for detailed annotation without resizing the main window.
  - **Right-click + drag** to pan across zoomed-in areas.
  - Non-destructive skipping: UI state and annotations persist across jumps.

Output Format

Each row represents a puck annotation at a given video timestamp, recorded in pixel coordinates relative to the video frame. This table is directly used as input to the rendering engine for simulating puck movement.

Table 4.3: Input: Example of Puck Annotation Output

Timestamp	Puck_X	Puck_Y
00:00:12.433	528	247
00:00:12.466	531	249
00:00:12.500	534	252

4.3.3 HockeySim Auto-Homography Tool

The HockeySim Auto-Homography Tool is a command-line utility that computes per-frame homography matrices aligning broadcast hockey footage to a standardized rink template. It is designed to operate on sequences of pre-extracted frames and output homography matrices in JSON format. Additionally, the tool can optionally generate overlay images for visual verification.

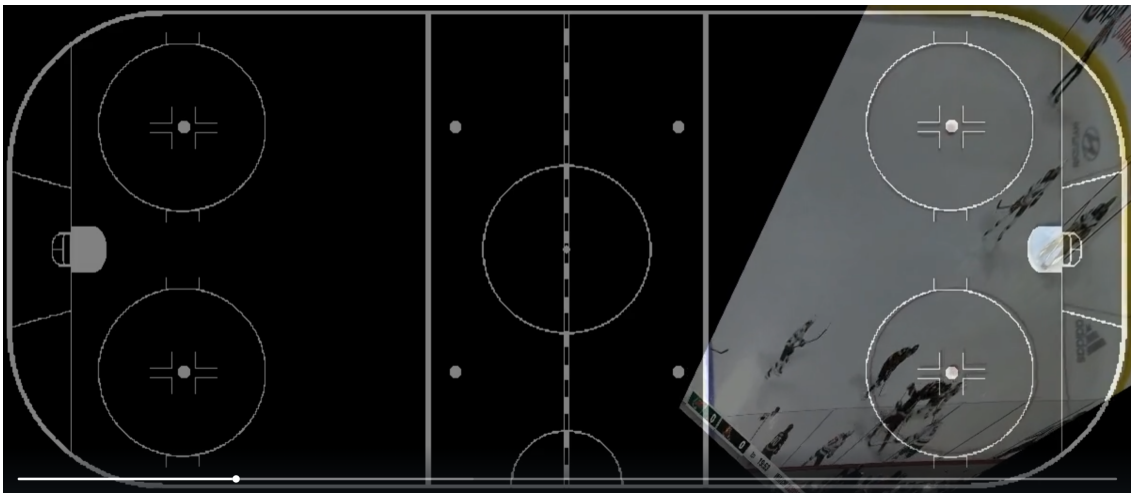


Figure 4.8: Example of debug overlay, where the computed homography has been applied to align the rink template with the broadcast frame.

The tool can be invoked via the following command:

```
python hockey_sim_auto_homography.py --input_dir <path>
                                      --output_json <file> [options]
```

It accepts the following arguments:

Table 4.4: Command-line arguments for the Auto-Homography Tool

Argument	Type	Description
`--input_dir`	str	Path to the directory containing input frame images (e.g., extracted video frames).
`--output_json`	str	Output path to save the computed homography matrix JSON file.
`--debug_dir`	str	(Optional) Directory to save debug overlay images with visual alignment projections.

Output Format

The tool outputs a JSON file containing per-frame homography matrices. Each matrix aligns the input frame to the standardized rink template. Below is a truncated example of the output format:

```
{
  "frame001.jpg": {
    "homography": [
      [0.58, 1.76, 126.07],
      [-0.13, 2.31, -230.09],
      [0.00004, 0.00257, 1.0]
    ]
  },
  "frame002.jpg": {
    "homography": [...]
  }
}
```

Each homography is a 3×3 matrix in row-major order, estimated from correspondences between the input frame and the template rink.

Debug Overlay Visualization

When `--debug_dir` is specified, the tool saves overlay images with visual projections of the rink template onto the original frames. This allows users to qualitatively assess the accuracy of the estimated transformations.

An example is shown in Figure 4.8, the background is the standard rink template (white lines on black), and the actual broadcast frame is overlaid onto it using the estimated homography. This overlay allows visual inspection of alignment accuracy, e.g., whether the projected frame correctly match the rink layout.

4.3.4 Player Tracking and Annotation Tool

This module performs automatic detection and annotation of player bounding boxes across a video sequence. The process relies on precomputed homography matrices to convert image-space detections into rink-space coordinates, aligning all player positions onto a shared spatial frame. The module is executed via a command-line interface and outputs a CSV file with per-frame player annotations.

CLI Example

```
python hockey_sim_player_track.py \
    --input_frames /path/to/frames \
    --homographies /path/to/homographies.json \
    --output_csv /path/to/output/players.csv \
    --confidence_threshold 0.8 \
    --model_checkpoint /path/to/model.ckpt
```

Parameter Descriptions

Table 4.5: Command-line arguments for the Player Tracking and Annotation Tool

Argument	Type	Description
`--video_path`	str	Path to the input video file from which frames will be extracted and processed.
`--homography_json`	str	Path to the JSON file containing precomputed per-frame homography matrices.
`--output_csv`	str	Output path where the annotated player tracking data will be saved in CSV format.
`--confidence_threshold`	float	(Optional) Minimum confidence score to include a player annotation. Defaults to 0.5.

Output Format

The tool generates a CSV file where each row corresponds to a single player instance in a given frame. Each row contains the following fields:

Table 4.6: Output CSV Format for Player Annotations

Frame	ID	X	Y	W	H	Conf	TeamID	RoleID	ActionID
0	1	709.81	442.05	83.02	147.18	0.93	-1	-1	-1
0	2	267.21	286.38	59.89	125.04	0.92	-1	-1	-1
...	...	...	...	...	...	...	...	...	...

Field Descriptions:

- **Frame**: Index of the frame within the sequence.
- **ID**: Assigned player ID (restarts each frame).
- **X, Y**: Top-left corner of the bounding box in rink coordinates.
- **W, H**: Width and height of the bounding box.
- **Conf**: Model confidence score for the detection.
- **TeamID, RoleID, ActionID**: Reserved fields for future classification extensions (e.g., team assignment, player role, or action label).

4.4 Unity Rendering Engine

The Unity Rendering Engine (Unity `6000.0.0f1`) is the final execution stage of the pipeline, responsible for synthesizing animated hockey gameplay scenes using structured input. This section documents how the engine works internally, the structure of its command-line interface, supported output formats, and animation logic.



Figure 4.9: Unity-based simulation environment showing scene preview and camera configuration prior to rendering.

4.4.1 Launch and Input Specification

The engine is launched via a Python wrapper script that interfaces with Unity's scene control and recording modules:

```
python hockeysim.py --input-mode tabular \
                    --game-state path/to/game_state.csv \
                    --player-meta path/to/player_metadata.csv \
                    --output path/to/output_directory \
                    --frame-rate 60 \
                    --resolution 1920x1080 \
                    --format mp4 \
                    --recording-name custom_video_name
```

Parameter Descriptions

The interface supports two modes:

Table 4.7: Command-line arguments for the Unity Rendering Wrapper

Argument	Type	Description
`--input-mode`	str	Input mode: either `tabular` (structured player/puck data) or `video` (reconstructed tracking output from broadcast footage).
`--game-state`	str	Path to the CSV file containing frame-by-frame player positions and sizes.
`--player-meta`	str	Path to the CSV file mapping player IDs to jersey numbers and teams. Required for `tabular` mode.
`--output`	str	Directory where the final rendered video (or image frames) will be saved.
`--recording-name`	str	(Optional) Custom name for the rendered output. If omitted, filenames are auto-generated.
`--frame-rate`	int	Frame rate of the rendered video. Default is 30.
`--resolution`	str	Output resolution, e.g., `1920x1080`. Default is HD.
`--format`	str	Output format: `mp4`, `avi`, `mov`, etc. Default is `mp4`.

- `tabular`: For synthetic gameplay generation using structured tracking data.
- `video`: For broadcast footage recreation using tracked players and annotated puck locations.

Once launched, Unity loads the required input data and enters interactive preview mode, allowing users to position the camera before rendering the scene.

4.4.2 Rendering Environment and Assets

The engine uses high-fidelity Unity assets for both the rink and animated players:

- Rink environment obtained from Tronitec Game Studios [56], featuring realistic boards, ice markings, and lighting. (see Figure 4.10)
- Player models with skating, idle, and shooting animations from Code This Lab [15]. (see Figure 4.11)

All players are placed according to the provided game-state data and are animated using motion-aware logic as described below.

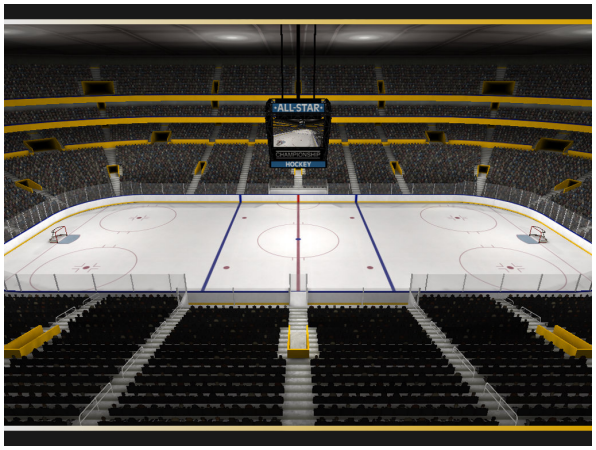


Figure 4.10: **(a)** Hockey rink asset. [56]



Figure 4.11: **(b)** 3D animated player asset with motion clips.[15]

4.4.3 Animation Logic

Directional Motion Blending

The animation controller infers skating direction and orientation dynamically. Let $\vec{v}_{\text{move}}$ be the velocity vector of a player between frames, and $\vec{v}_{\text{face}}$ be the unit vector pointing toward the puck. The relative angle θ between them determines the animation state:

- $|\theta| < 45°$: forward skating
- $45° \leq |\theta| \leq 135°$: side-step skating
- $|\theta| > 135°$: backward skating

Shot Animation Triggering

To realistically simulate shooting behavior without requiring manual shot annotations, the system employs a rule-based heuristic to trigger shot animations based on puck dynamics and player proximity.

Let $p_t \in \mathbb{R}^2$ denote the puck position in image-space coordinates (pixels) at frame t, and let $\vec{v}_p(t) = p_{t+1} - p_t$ represent the puck's frame-to-frame motion vector. A shot event is inferred at frame t if the following conditions are jointly satisfied:

1. **Velocity spike:**

$$\|\vec{v}_p(t)\| > \tau_v$$

where $\tau_v = 25$ pixels is the velocity threshold, corresponding to a sudden puck displacement of at least 25 pixels between two consecutive frames.

2. **Player proximity:** There exists a player i such that:

$$\|p_t - P_t^i\| < \tau_d$$

where P_t^i is the image-space position of player i at frame t, and τ_d is a configurable spatial proximity threshold (e.g., 20 pixels).

3. **Directional alignment:**

$$\cos\theta = \frac{(p_{t+1} - p_t) \cdot (p_t - P_t^i)}{\|p_{t+1} - p_t\| \cdot \|p_t - P_t^i\|} > \tau_\theta$$

where τ_θ (e.g., 0.9) is a cosine similarity threshold, ensuring the puck's motion is directed away from the nearby player.

If all three conditions are satisfied, the system triggers a shot animation for player i at frame t. This rule-based strategy enables realistic visual feedback in the Unity simulation without requiring manual annotation of shot events or complex physical modeling.

This logic enables visually plausible shot behaviors in the simulation without the need for explicit event labels or fine-grained interaction data.

Homography Smoothing

To ensure temporal stability in rendered player trajectories, a homography-aware smoothing algorithm is applied to the player positions prior to rendering. This process mitigates frame-to-frame jitter introduced by noise in detections or minor homography inconsistencies.

Let $P_t = (x_t, y_t)$ denote the player position in the canonical rink coordinate system at timestamp t, as derived via homographic projection from the original frame. The goal is to compute a smoothed trajectory $\tilde{P}_t = (\tilde{x}_t, \tilde{y}_t)$ that preserves the original motion trend while suppressing high-frequency noise.

We use a weighted temporal filter of the form:

$$\tilde{P}_t = \frac{1}{Z} \sum_{k=-N}^{N} w_k \cdot P_{t+k} \quad \text{where} \quad Z = \sum_{k=-N}^{N} w_k$$

Here, w_k is a symmetric Gaussian kernel (e.g., $w_k = e^{-\alpha|k|}$), and $2N+1$ is the smoothing window size $N = 3$. This filtering is applied independently to both x and y coordinates. If frames near the boundary are unavailable (e.g., $t + k$ is out of bounds), the filter is adaptively renormalized.

To ensure geometric consistency, all positions P_{t+k} are assumed to lie in the rink coordinate system (i.e., post-homography transformation). This avoids blending errors that would arise if the smoothing were performed directly in pixel space on distorted frame coordinates.

4.4.4 Synchronized Annotation Output

In addition to the rendered gameplay videos, our tool automatically exports structured annotations aligned to each frame of the simulation. These annotations enable direct use of the synthetic data for computer vision training, benchmarking, and scene understanding tasks.

The following auxiliary outputs are generated:

- **Player Bounding Boxes:** For each rendered frame, the 2D image-space bounding boxes of all players are saved to a CSV file. This includes player IDs and team affiliation metadata. (see Table 4.8)
- **Player Rink Coordinates:** All players' (X, Y) positions in the canonical rink coordinate system are exported in parallel with each frame, enabling motion reconstruction and strategy analysis. (see Table 4.10)
- **Segmentation Masks:** High-resolution binary segmentation masks are generated for each player per frame. These are saved as individual PNG images in a structured directory format. (see Figure 4.12)
- **Frame Metadata:** Frame-wise timestamps, player IDs, team designations, and full camera parameters are included in supplemental metadata files. (see Tables 4.9, 4.10)

Output Samples

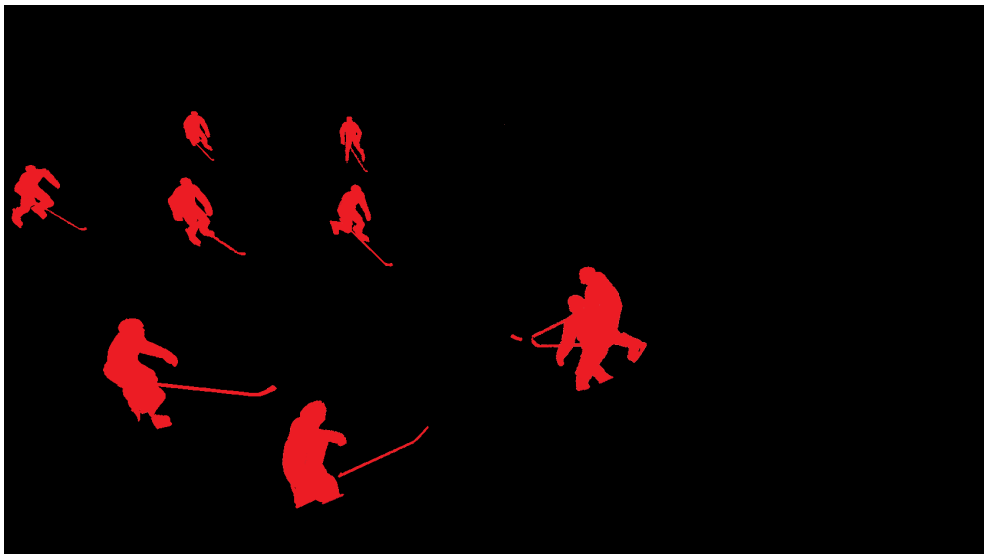


Figure 4.12: Example of segmentation mask of a rendered frame.

Table 4.8: Example bounding box table (pixel-space)

Frame	Player ID	X_min	Y_min	X_max	Y_max
frame_0001	P1	220	340	280	420
frame_0001	P2	300	360	360	430
frame_0002	P1	225	345	285	425
frame_0002	P2	305	362	365	432

Table 4.9: Example frame metadata including full camera parameters

Frame	Timestamp (s)	Camera Mode	FOV (deg)	Cam X	Cam Y	Cam Z	Rot X (deg)	Rot Y (deg)	Rot Z (deg)	Near/Far
frame_0001	0.000	dynamic	60.0	0.0	25.0	15.0	–	–	–	–
frame_0002	0.033	dynamic	60.0	0.1	25.0	15.1	–	–	–	–
frame_0003	–	–	60.0	-0.37	22.70	-30.46	31.1	0.4	0.0	0.3 / 1000

Table 4.10: Example player positions in rink coordinate system

Frame	Player ID	X_rink	Y_rink
frame_0001	P1	24.3	12.1
frame_0001	P2	33.7	17.9
frame_0002	P1	24.9	12.4
frame_0002	P2	34.2	18.1

Output Folder Structure

All outputs are organized into a timestamped directory (e.g., `2025-08-05_17-42-33_rendered_game/`) containing the following structure:

```
/output_root/
 frames/                 # RGB video frames (PNG)
 video/                  # Final rendered video (e.g., .mp4)
 bounding_boxes.csv      # Image-space player bounding boxes
 rink_coordinates.csv    # Canonical rink-space positions
 player_metadata.csv     # Jersey + team info
 frame_metadata.csv      # Timestamps, camera settings
 masks/                  # Instance segmentation masks (PNG)
 logs/                   # Render logs and config
```

4.4.5 Camera Control and Output Options

Unity supports both fixed and dynamic cameras:

- **Fixed**: Mimics broadcast-style top-down or side views.
- **Dynamic**: Lets user alter camera parameters during the recording.

Output options include:

- Resolution: up to 4K
- Frame rate: up to 120 FPS
- Formats: MP4 (H.264), MOV, AVI, WebM
- Recording name: auto-generated if not specified

4.5 Usage Workflow

To provide a practical view of how the tool is used in real-world settings, we describe two representative usage scenarios: **(1)** generating synthetic gameplay scenes from symbolic data, and **(2)** recreating real-world gameplay from broadcast video. Each workflow is illustrated with command-line examples, detailed parameter guidance, and a description of expected outputs.

4.5.1 Scenario 1: Scene-Generating Data Augmentation from Tabular Input

This scenario applies when the user has access to structured tracking data of player and puck movements, either generated by simulation tools or extracted from analytics platforms. One such example is **Franchise Hockey Manager 11 (FHM11)**[1], an NHL-licensed hockey strategy simulator that represents players and puck as 2D dots. The game engine can replay historical seasons or simulate entirely new matches, and in both cases it produces synthetic trajectories of players and the puck (not reconstructed from broadcast video). These time-stamped positions can then be exported and used to synthesize realistic 3D gameplay animations.

[1] https://store.steampowered.com/app/2936410/Franchise_Hockey_Manager_11/

Step-by-Step Workflow

1. **Prepare Input Tables:**

   Format the extracted data into:

   - A `game_state.csv` file with frame-by-frame player and puck positions (see Table 4.1).
   - A `player_metadata.csv` file mapping player IDs to jersey numbers and teams (see Table 4.2).

2. **Launch the Simulation:** This will launch Unity in interactive mode to allow fine-tuning of camera and recording settings:

```
python hockeysim.py --input-mode tabular \
                    --game-state path/to/game_state.csv \
                    --output path/to/output_directory \
```

3. **Unity Scene Rendering and Export:**

   The system uses the specified settings to animate the gameplay. Once satisfied with the camera placement, users can initiate recording from the Unity interface. The output video will be saved with the specified resolution, format, and frame rate.

4.5.2 Scenario 2: Recreating Broadcast Footage from Real Game Video

This scenario covers the case where only a raw broadcast or fan-recorded video is available. The system enables the user to recreate the gameplay in a standardized, controllable 3D environment using visual processing and optional manual annotations.

Step-by-Step Workflow

1. **Extract Frames:**

   Pre-process the video using `ffmpeg`:

```
ffmpeg -i raw_game.mp4 frames/frame_%04d.png
```

2. **Homography Estimation:**

   Depending on the video type:

   - **Fixed camera:** Launch the manual homography tool (GUI-based).

     ```
     python hockey_sim_manual_homography.py
     ```

   - **Moving camera:** Use the automated homography estimator.

     ```
     python hockey_sim_auto_homography.py --input frames/
                                          --output homographies/
     ```

3. **Run Tracking and Annotation:**

   ```
   python hockey_sim_player_track.py --input frames/ \
                                     --homography homography.json \
                                     --output tracked_players.csv
   ```

4. **Annotate puck positions** (GUI-based):

   ```
   python hockey_sim_puck_annotate.py
   ```

5. **Render Scene:** (GUI-based)

   ```
   python hockeysim.py --input-mode video \
                       --game-state tracked_players.csv \
                       --output output_video/ \
   ```

   This launches Unity with pre-loaded tracking and puck data. Users can adjust camera views before starting the recording (see Figure 4.9).

6. **Output:**

   A rendered 3D animation of the original game is generated with accurate player movement and puck behavior, enabling replays or visual analytics from any camera angle.

4.6 Rendering Examples

To illustrate the full pipeline in use, this section presents representative frames rendered by the Unity engine under four camera configurations: a fixed *top-down* view, a fixed *side* view, a *dynamic follow* camera, and a *broadcast-like* camera. These examples are generated from the same input sequence to highlight how viewpoint choice affects depth cues, occlusion ordering, and perceived motion (Figures 4.13, 4.14, 4.15, and 4.16). The goal is not only to show visual output, but also to convey how camera parameters expose different aspects of the play for downstream analytics (e.g., detection, tracking, pose, and event recognition).

Looking ahead. The Unity-based simulator presented here extends augmentation from isolated objects to full-scene reconstruction, providing controllable multi-view data and rich annotations for hockey analytics. The next chapter, Chapter 5, compares this scene-level simulation with the object-level SLDA approach, and discusses their complementary roles and combined use in constructing robust training datasets.

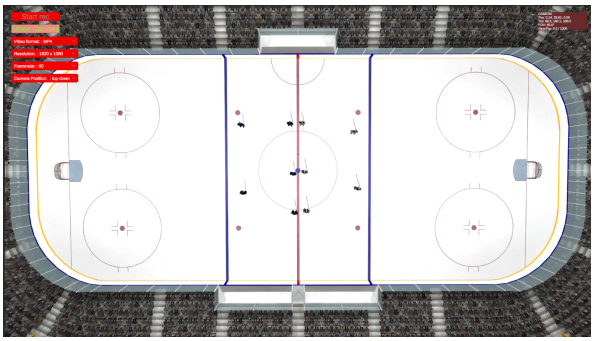


Figure 4.13: **(a)** Simulation settings for a fixed top-down camera.

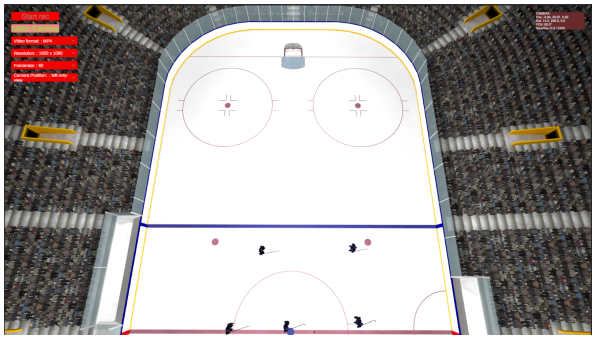


Figure 4.14: **(b)** Simulation settings for a fixed side-view camera.



Figure 4.15: **(c)** Simulation rendering result for a dynamic camera.



Figure 4.16: **(d)** Simulation rendering result for a broadcast-like camera.

Chapter 5

Comparative Analysis of SLDA and the Unity Simulation Tool

With object-level augmentation (Chapter 3) and scene-level simulation (Chapter 4) established, this chapter compares their goals, capabilities, and tradeoffs. We contrast augmentation within real images against full-scene synthesis, analyze training impacts and domain gaps, and outline where each method is most effective on its own and in combination.

5.1 Addressing Data Scarcity and Complexity in Hockey Analytics

In this thesis, we tackled two interrelated challenges in hockey analytics: the lack of sufficient labeled data for robust hockey stick detection/segmentation, and the difficulty of bridging abstract hockey game data with rich visual analysis. We approached these problems with two distinct yet complementary methods. The first method, Synthetic Local Data Augmentation (SLDA), addresses data scarcity by augmenting existing 2D images with realistic synthetic examples of the target object (hockey sticks). The second method, the Unity-based 3D Hockey Simulation Tool, creates completely new hockey scenes in a virtual 3D environment, turning symbolic game data into synthetic video, thereby enriching the available visual data and enabling novel analysis perspectives. Both approaches leverage synthetic data generation to overcome real-data limitations, but they operate at different levels (local 2D augmentation vs. full-scene 3D simulation) and serve complementary purposes. Together, they contribute to a larger vision of using simulation and

augmentation to advance sports analytics.

SLDA's Role: SLDA was designed to improve model training when real training data is limited or lacks diversity. By generating additional annotated images with augmented hockey sticks, SLDA directly increases the volume and variety of training examples available to a stick segmentation model. Crucially, SLDA's augmentations are context-aware: sticks are inserted into real game images in plausible positions and with photorealistic distortions (motion blur, rotations, etc.), as discussed in Chapter 3. This addresses the scarcity of rare but important cases, for example, images of partially occluded or motion-blurred sticks, which are under-represented in the raw dataset. By synthetically creating these cases, SLDA helps the model learn to handle them, thereby boosting robustness under adverse conditions. We saw this reflected in the higher recall and F1 scores for SLDA-augmented models, especially on blurry sticks that baseline models often missed. In summary, SLDA tackles the data limitation problem by augmenting what data we do have, it works within the paradigm of supervised learning on 2D images, making the most of scarce annotations through synthetic augmentation.

Unity Simulation Tool's Role: The Unity-based 3D Hockey Simulation Tool approach addresses data scarcity and complexity from a different angle: instead of augmenting existing images, it generates entirely new synthetic visual data from underlying game descriptors. This tool takes in structured information (e.g., player and puck coordinates over time, or trajectories extracted from a video) and reconstructs full hockey scenes in a 3D virtual rink. By doing so, it creates an unlimited supply of annotated video frames: every rendered frame from the simulation comes with known ground-truth for object positions, identities, etc., by virtue of the simulation's internal state. For example, if we input actual player/puck tracking data from a game, the tool can output a re-enacted sequence from any camera angle, complete with perfect labels for each player and the puck at each frame. This approach is extremely valuable for generating data for tasks beyond stick segmentation: e.g., training or evaluating player trackers, multi-view action recognition models, or tactics analysis, where large amounts of annotated video are needed. Moreover, the simulation addresses complexity by bridging between abstract data and visuals - it allows researchers to visualize and analyze game dynamics (player movements, interactions) in ways not possible with a single real camera feed. It provides a sandbox to explore "what-if" scenarios or replay sequences from various perspectives, aiding qualitative understanding in addition to producing training data.

In summary, both SLDA and the Unity-based 3D Hockey Simulation Tool combat the lack of data, but at different scales: SLDA augments within real images to bolster a specific computer vision model, whereas the simulation generates entire new images/videos to broaden the dataset or to inspect game situations. SLDA primarily focuses on object-

level augmentation (hockey sticks) to improve detection/segmentation accuracy, while the simulation tackles scene-level synthesis, creating complex scenes with multiple entities for tasks such as tracking, pose estimation, or strategy analysis. Both methods enhance model training - SLDA by improving model robustness on real images, and the simulation by supplying additional synthetic training examples at scale, thus each plays a role in overcoming data scarcity and complexity in hockey analytics.

5.2 Data Augmentation vs. Synthetic Simulation for Model Training

SLDA integrates directly into the model training pipeline as a data preprocessing step. The augmentation happens on-the-fly (or in a pre-processing stage) to expand the training dataset with modified versions of existing images. One of SLDA's strengths is that it leverages real image backgrounds combined with real object cutouts, preserving high realism. Because the inserted sticks come from real photographs and are placed in real scenes, the augmented images maintain the complex textures, lighting, and context of actual game footage. This likely contributed to the improved performance we observed, the model sees a greater range of stick appearances (different angles, blur levels, etc.) without being thrown off by unnatural artifacts.

SLDA's effect is to enrich the training distribution: for instance, in Chapter 3 we noted that SLDA alone boosted mAP@50 to 0.895 vs. 0.842 baseline, implying the model learned to detect sticks it previously could not. The augmentation specifically targeted the model's weaknesses (e.g., motion blur), resulting in a more robust detector.

However, SLDA's scope is somewhat narrow: it augments one object class (sticks) in existing images. It does not introduce new types of data or fundamentally new scenarios beyond what's in the original footage, instead, it locally perturbs and inserts within those scenes. For training a stick segmentation model, this is exactly what we needed. But for other tasks (say, identifying player numbers or recognizing actions), SLDA would not directly provide new training scenarios except for what can be done via copy-pasting relevant objects.

SLDA is best viewed as a targeted data augmentation to address specific dataset deficiencies (in our case, lack of blurred/occluded stick examples). It fits seamlessly with supervised learning: one can turn it on during training to boost performance without altering the model architecture or loss function. The cost is relatively low, SLDA augmen-

tation added only ∼0.24s per image in processing overhead, which is minor in a training pipeline. Thus, SLDA is a lightweight, focused method to get more out of a limited dataset.

The Unity-based simulation offers a more expansive way to generate training data. By recreating entire scenes, we are not limited to augmenting what the camera saw, we can simulate scenarios that never occurred in our footage, or generate views from angles that were never filmed. For example, we can simulate a power-play setup or a particular defensive formation and get thousands of labeled frames from arbitrary camera perspectives. This is extremely powerful for training models that require diverse data.

In principle, one could use the simulation to create a synthetic training dataset from scratch. For instance, to train a player detection model, we might simulate hundreds of games with random team positions, varied camera moves, etc., and then train the detector purely on rendered frames (with perfect labels from the simulation). The simulation tool provides full control over the data generation process: we can adjust lighting, camera angles, jersey appearances, and even define the motions of players and puck if needed.

This level of control enables generating corner cases or rare events in abundance (e.g., unusual camera angles, or scenarios such as six players crowding the net) which a model might otherwise rarely see.

One key difference is that simulation-generated data is fully synthetic. Even with a high-quality game engine, there may be a domain gap between rendered frames and real broadcast footage. Textures, lighting, or motion in the simulation might not perfectly match reality, potentially limiting how well a model trained on simulation transfers to real video. SLDA, by contrast, works within real images, so the augmented data retains real image characteristics by design.

In our use, we treated the simulation more as a means for analysis and generating supplemental data rather than the sole training source. For example, the Unity tool can produce multi-view videos of the same play, which is useful for tasks such as multi-camera action recognition or 3D pose estimation, tasks where obtaining real multi-view data is logistically difficult. A model trained on such synthetic multi-view data could then be fine-tuned on limited real data. Similarly, the simulation could generate labeled trajectories to train a tracking algorithm (with the ground truth positions known, one can supervise the tracker effectively).

In our context of hockey stick segmentation, the simulation tool could be used to supply additional training images of sticks, but with a different flavor than SLDA. For instance, we could simulate a sequence and then extract frames where sticks are visible, obtaining synthetic stick images in various poses. However, unless the visual fidelity is extremely

high, a segmentation model might not benefit as much from those as from SLDA images that blend real and synthetic.

On the other hand, the simulation could incorporate a stick segmentation model (trained via SLDA) to automatically annotate sticks in the simulated videos, to validate that the simulation's output is analyzable by current models. This highlights a complementary use: using the SLDA-trained model to analyze or label simulation-generated data. Because the SLDA model is robust, it might successfully detect sticks even in the slightly off-domain simulation frames, thus providing a way to evaluate or refine the simulation (e.g., checking if a simulated scene is realistic enough that a real-trained model can detect sticks in it).

In summary, SLDA vs. Simulation for model training can be seen as augmentation versus synthesis. SLDA augments existing data to directly improve a model's training process, which we validated with clear performance metrics. The Unity-based 3D Hockey Simulation Tool synthesizes new data which can either be used to train models for tasks beyond what our original data could support, or to evaluate models in novel scenarios. Each has its place: if one's goal is to push a single-model performance on a known task (such as stick detection), SLDA is highly effective and simple to deploy. If the goal is to create an entirely new capability (say, a system that tracks all players and the puck in 3D, or an AI referee to recognize events), the simulation provides the playground to generate and test those complex scenarios.

5.3 Interoperability and Complementary Use of Both Methods

While SLDA and the Unity-based 3D Hockey Simulation Tool were developed independently to tackle different facets of our problem, they can in fact complement each other in a larger pipeline. There are several points of interoperability:

- *Simulation Feeding Augmentation:* The Unity tool can generate synthetic images or sequences that SLDA could further augment. For example, one could render a series of hockey scenes in Unity (perhaps replicating various play situations) and then apply SLDA on those rendered images to insert additional sticks or augment existing ones. This would combine the benefits of both: we'd have fully synthetic scenes (with known ground truth for all objects) and within those, extra augmented variations of certain objects. In practice, this could be useful if the simulation by itself doesn't produce enough motion blur, SLDA could inject blurred stick instances into

the simulated frames to mimic camera motion blur or fast puck movement that might be hard to simulate physically. By doing so, the synthetic dataset from Unity would become even more diverse and challenging, potentially improving the training of vision models. Essentially, the simulation can provide new backgrounds and contexts (e.g., different arena layouts, camera viewpoints), and SLDA can ensure that within those contexts, critical objects (sticks) appear with the variety of conditions needed (blurs, occlusions).

- *Augmentation Aiding Simulation Analysis:* Conversely, SLDA-augmented models can be applied to analyze simulation outputs. Since the simulation can replay any scenario, one could use a model (say, a stick detector or player tracker) on both real and simulated videos. If the model is trained with SLDA on real data, it might be robust enough to handle some domain differences in simulation output. Using the same model on simulation allows us to validate if simulation frames are realistic: for instance, does a stick segmentation model trained on real footage also perform well on a simulated video of a game? If yes, that increases confidence that the simulation is visually faithful; if not, it might indicate areas to improve in the renderer (e.g., lighting or motion realism). Also, applying models to simulation can generate automatic annotations for the synthetic sequences. As an example, suppose we want to label when a stick is lifted above shoulder level in a simulated game (a high-sticking event), an SLDA-trained stick detector could process each frame of the simulation to find stick masks, and then the simulation's known camera calibration could determine the stick's height. In this way, a combination of simulation and model can yield analytics that would be difficult to get from real video alone (where ground truth is unknown).

- *Unified Training Pipeline:* Looking ahead, one can imagine a unified pipeline where simulation and augmentation are used together to train a suite of models for hockey analysis. The pipeline might start with real data (limited but valuable) and a simulation engine (to generate unlimited but synthetic data). SLDA could be applied to the real data to improve a particular detection model (as we did for sticks). Meanwhile, the simulation could be used to pre-train models for tasks where real data is extremely scarce (e.g., predicting player trajectories or recognizing set plays). The models pre-trained on simulation could then be fine-tuned on real data augmented via SLDA.

  This way, simulation provides breadth and SLDA provides focus. For instance, a player pose estimation model might be pre-trained on thousands of simulated player images (with ground truth joint positions), but since simulated players might not

look exactly like real ones, we could fine-tune that model on a smaller set of real hockey images augmented by SLDA (if we had segmented body parts, etc., to insert or augment). Although we did not explicitly do this in our thesis, the principle stands: SLDA and simulation are not mutually exclusive; together they can form a multi-stage data generation strategy.

- *Hand-off via Data Formats:* Practically, the integration could be facilitated by common data formats. Our Unity tool outputs annotations such as a "game-state table" (players and puck coordinates per frame) and can also output rendered images or video. These outputs could be fed into a data augmentation pipeline (like a Python script using our SLDA code) as if they were real images. Conversely, one could take real tracked data (output from a vision model on real video) and feed it into the simulation to reconstruct the play.

  The interplay means improvements in one component can feed the other: e.g., better homography and player tracking from the vision side (possibly enhanced by data augmentation) leads to better inputs for simulation recreation; better simulation outputs lead to better synthetic data for training vision models, and so on. Thus, we see a potential feedback loop where simulation and augmentation jointly improve the overall analytics pipeline.

In our work, the two methods were applied to separate sub-problems (SLDA for stick segmentation, Unity tool for scene generation and visualization). But the philosophy behind them is similar, using synthetic data to overcome real-world data limitations, and they naturally dovetail. A complete hockey analytics system in the future could use simulation to generate scenarios at scale, SLDA (or similar augmentations) to fine-tune models to real-world nuances, and those models in turn to analyze both real and simulated hockey data. This complementary use would maximize the strengths of each approach.

5.4 Strengths, Limitations, and Use-Case Tradeoffs

Both SLDA and the Unity-based 3D Hockey Simulation Tool come with unique strengths and some limitations, which often make them suited for different use-cases in hockey analytics:

SLDA Strengths

(a) Improved Model Performance: As demonstrated, SLDA yields tangible accuracy gains for object segmentation/detection tasks. It directly targets known weaknesses (e.g., motion blur) in the model's training data, leading to higher recall and better stability.

(b) Simplicity and Low Cost: Implementing SLDA is relatively straightforward in a training pipeline and does not require complex infrastructure, a Python script can generate augmented images on the fly. The computation is light (a few image operations per augmentation).

(c) Realism: By using actual images and segmentations, the augmented outputs remain photo-realistic. We insert real sticks into real backgrounds, avoiding the simulator's potential uncanny artifacts. This likely contributed to seamless generalization from augmented training data to real test data.

(d) Fine-Grained Control: SLDA allows per-instance customization. For example, one could choose to only augment sticks of a certain size or only apply blur in some proportion of images, etc. This fine control means we can tailor augmentation to the specific data distribution. In our case, we knew motion blur was a major issue, so we set a high probability for that transform (87.5%).

SLDA Limitations

(a) Object-Specific: SLDA requires a segmented object dataset (we had to manually collect 209 stick images). For each new object class to augment, one would need to curate a similar catalog. It's not easily scalable to augmenting all objects in a scene, we focused on sticks.

(b) Context Constraints: We maintained context by inserting sticks in logical places (on ice). If the original data lacks certain contexts (e.g., no images of sticks near the bench), SLDA won't invent those fundamentally new contexts, it still works within the realm of the original scenes.

(c) No New Scene Content: SLDA doesn't create new backgrounds or new players, it can't simulate an entirely new game situation that wasn't present. For example, if our real dataset had no images of an empty rink, SLDA can't generate an empty rink scenario with sticks; it always adds to existing images.

(d) Diminishing Returns: In extreme cases, if one augments too heavily, the model might start seeing unrealistic images (e.g., too many sticks in odd places). We combined SLDA with traditional augmentation and saw further gains, but one must be careful not to break the realism or the underlying data distribution too much.

Unity Simulation Strengths

(a) Unlimited Data Generation: The simulation can produce essentially infinite examples. One can vary game situations, generate full games worth of frames, or systematically sweep through camera angles. This is invaluable for data-hungry deep learning models or for covering edge cases.

(b) Full Ground Truth: Unlike real data, simulated data comes with automatic labels for every entity (each player's identity and position, puck position, etc.) at every frame. This enables supervised training for tasks where labeling real video is extremely labor-intensive (such as tracking multiple objects or 3D pose estimation).

(c) Scenario Reproduction and Beyond: The tool can recreate specific real sequences (by inputting tracked coordinates), this is useful for visual analysis (coaches can replay a play in 3D and view from any angle). It can also produce hypothetical scenarios (what-if situations) by altering the input data.

(d) Multi-modal Analytics: Because it unifies spatial data and visual output, the simulation can be used to test computer vision algorithms under known conditions or to generate synthetic experiments (e.g., gradually increasing crowd noise or changing lighting in simulation to see when a vision model fails). It's a versatile research platform beyond just data generation.

Unity Simulation Limitations

(a) Visual Realism Gap: Creating a perfectly realistic hockey simulation is challenging. Our tool, while high-resolution, may not capture all nuances of, say, jersey cloth simulation, skate marks on ice, or crowd reflections. Such differences can make models trained on simulation less effective on real data unless domain adaptation techniques are used.

(b) Development Complexity: Building and maintaining the simulation required integrating multiple components (Unity for rendering, Python for data parsing, homography for alignment, etc.). It's a complex software engineering task, which is justified for broad research but might be overkill for a narrow problem.

(c) Real Data Dependency: If we want to simulate real games accurately, we need good input data (tracking, homography from the real video). Errors in those inputs (such as player tracking mistakes) will carry over to the simulation. So the simulation doesn't eliminate the need for vision; rather, it complements it.

(d) Computational Cost: Rendering thousands of frames in Unity with high fidelity and then processing them is more computationally intensive than SLDA's quick image augments. While it's feasible (and can be parallelized), generating a large synthetic dataset

might take significant time and storage (for video frames), whereas SLDA could generate on the fly without storing anything long-term.

Use-Case Scenarios Given these tradeoffs, each method excels in different scenarios. If the goal is to improve a specific vision model's accuracy on a task where data is limited (such as our stick segmentation), SLDA or similar augmentation is a first-line solution, it's easy to implement and directly boosts performance, as we saw with metrics going up by several points. On the other hand, if the goal is to explore new analytics questions or generate data for an entirely new model (such as training a player pose estimator or testing a multi-view action recognition approach), the simulation tool is invaluable. It provides a means to get training data or to validate algorithms in a controlled environment.

For example, consider player tracking: Labeling player identities and trajectories in video is tedious, but the simulation can output perfect tracks if we input the data (or even simulate players moving in typical patterns). A tracking algorithm could be trained on these and then fine-tuned on a small real dataset. In this use-case, SLDA would be less directly applicable, because the problem is not a lack of appearance examples, but rather a lack of any ground-truth for trajectories, something only the simulation can fully supply. Another scenario: 3D pose estimation of players (estimating joint angles of players from video). A possible approach is to use simulated 3D player models in Unity to generate synthetic images with ground-truth 3D joint positions. SLDA alone cannot generate new poses that were never in the data; it could augment players somewhat (e.g., copy a player into another image, maybe with slight rotation), but it can't easily create new human poses. The simulation, however, could animate a 3D skeleton in arbitrary poses and render it. So for pose estimation, simulation data would be key, possibly combined with augmentation to adapt to real images.

Finally, there's the question of interactivity and visualization: the Unity tool shines in situations where a human analyst wants to visualize a play or when one needs to demonstrate something (such as a "virtual replay"). SLDA doesn't offer any interactive visualization, it's purely data for machine consumption. So for any use-case involving coaches, broadcast enhancements, or user-facing analytics, the simulation is the method of choice. It also serves educational and analytical purposes, e.g., a coach can replay sequences from any perspective to illustrate a point, as noted in Chapter 4. In contrast, SLDA's output is not meant to be viewed by humans (augmented training images might even look odd with multiple sticks), it's meant to improve a model's internal understanding.

In conclusion, SLDA and the Unity-based 3D Hockey Simulation Tool each have domains where they are most beneficial. SLDA is a targeted, efficient technique to enhance model training on specific detection/segmentation tasks with limited data, ensuring mod-

els can handle challenging visual effects such as motion blur. The Unity-based 3D Hockey Simulation Tool is a broad, generative approach that provides a platform for creating and analyzing hockey game data in ways not possible from the original video alone. The trade-off often comes down to specificity vs. generality: SLDA is very specific (hockey sticks in 2D images), whereas simulation is general (any object in a 3D scene but with less focus on one particular detail). Depending on the problem at hand, one or the other (or both in tandem) will be the appropriate tool in the arsenal for advancing hockey analytics.

Looking ahead. The complementary roles of SLDA and the Unity simulation tool suggest a broader strategy for advancing hockey analytics. SLDA offers a lightweight, targeted way to strengthen model robustness within real images, while the simulation tool provides scalable generation of annotated scenes that extend beyond the limits of available footage. Together, they illustrate how object-level augmentation and scene-level synthesis can be combined to mitigate data scarcity and address analytic complexity. Future systems may build on this interplay, using augmentation to refine model sensitivity to critical details and simulation to supply breadth across scenarios, perspectives, and tasks. In Chapter 6, we return to this theme to outline the broader opportunities such integration opens for computer vision in sports analytics.

Chapter 6

Conclusions and Future Work

Building on the comparative perspective developed in Chapter 5, this final chapter draws together the key contributions of the thesis and places them within the broader context of hockey video analytics. We begin with a summary of the main outcomes of SLDA and the Unity simulation tool, and then outline promising directions for future research motivated by their strengths and limitations.

6.1 Summary of Contributions

SLDA – Improving Hockey Stick Segmentation We introduced Synthetic Local Data Augmentation as a novel strategy to boost object segmentation performance when annotated data is limited. By injecting realistic synthetic hockey stick instances into real images, SLDA significantly increased the diversity of training data without requiring new manual annotations. Key innovations included using a custom dataset of real stick segmentations as insertion objects, applying context-aware transformations (especially motion blur) to simulate challenging conditions, and intelligent placement of sticks in semantically appropriate locations (e.g., on the ice surface). In a case study on hockey stick mask segmentation, SLDA led to substantial accuracy gains: our YOLOv8-seg model trained with SLDA achieved higher precision, recall, and F1-score than the same model trained on original data or with conventional augmentations. Quantitatively, we saw improvements such as +5.8% in mAP@50 and +4.0% in F1-score over the best baseline (Chapter 3). Qualitatively, the SLDA-augmented model was uniquely capable of detecting sticks under heavy motion blur and partial occlusion, scenarios where other models failed. This

confirms that targeted synthetic augmentation can effectively compensate for missing examples in the training set, making the model more robust to real-world challenges. The SLDA methodology is generalizable to other instances and sports (e.g., augmenting soccer balls, etc.), and this work provides a blueprint for how carefully crafted augmentations can yield outsized benefits in performance. Furthermore, as part of this contribution, we built a custom annotated hockey stick dataset (209 unique stick images, and a training set of 1000 hockey frames with 4226 labeled sticks) which has been made available for the community. This dataset itself is a practical asset, addressing the lack of public data.

Unity-Based Simulation Tool – Bridging Data and Vision We developed a comprehensive 3D simulation platform that converts abstract hockey game data into realistic visual sequences. The tool allows users to input either tabular player/puck coordinates or broadcast video (with detection/tracking), and outputs a reconstructed hockey game in a Unity 3D environment, complete with moving player models, a puck, and a configurable virtual camera. This contribution serves multiple purposes. For model development, the simulation can generate synthetic video data with ground-truth annotations at a scale and variety far beyond the original data. For example, it can produce multi-angle video of the same play, which we discussed as a way to train and evaluate algorithms for multi-view player tracking or action recognition. It can also create entirely new scenarios (e.g., simulating a power play or testing camera viewpoints), contributing to data augmentation in a broader sense (Chapter 5). We emphasized modularity in the tool's design (Chapter 4): it includes a homography alignment module to map real video to the rink coordinate system, a player detection & tracking module (leveraging an existing multi-object tracking approach) to get players' trajectories from video, and an integrated rendering engine in Unity that takes the processed data and generates a video output. By uniting these components, we created a pipeline that can take raw game footage and produce a faithful 3D reenactment. This is a step toward bridging the gap between data and understanding: rather than analyzing CSV files of coordinates or single-camera videos in isolation, researchers and practitioners can now move fluidly between the abstract data and a visual, immersive representation of the game. The tool's utility was demonstrated in enabling synthetic data generation (e.g., for training a model or testing performance under known conditions) and visual analytics (e.g., verifying a tracking algorithm by visualizing its output in 3D, or illustrating a player's positioning to a coaching staff). This contribution lays a foundation for more interactive and extensive uses of simulation in sports analytics.

Combined Impact Both contributions target the core thesis objective of overcoming limitations in hockey video analytics, one by augmenting reality with synthetic data, the other by creating synthetic realities from data. The SLDA technique directly improved the performance of a vision model critical for understanding hockey gameplay (identifying

sticks, which relate to player actions and penalties). The Unity-based 3D Hockey Simulation Tool expanded the horizons of what we can do with hockey data, enabling experiments and visualizations not possible with the raw video alone. Together, they represent a cohesive advancement in using synthetic data as a powerful tool in sports engineering: SLDA showed that even simple synthetic additions can make machine learning models more robust, and the simulation demonstrated that entire synthetic worlds can be built to support analysis and training. The synergy between them was discussed in Chapter 5, highlighting how a robust model (trained via SLDA) and a realistic simulator can be combined for a virtuous cycle of improvement and analysis. Overall, this thesis contributes to the field by: (1) introducing a new data augmentation method (with demonstrated efficacy in a peer-reviewed study), (2) providing a novel simulation platform tailored to hockey. While this implementations focused on hockey stick segmentation and game reenactment, the underlying ideas are extensible to other sports and domains where data is scarce or fragmented.

6.2 Future Work

The work presented in this thesis opens several avenues for future exploration and improvement. Below, we outline concrete directions for future research and development, inspired by both the limitations we encountered and suggestions from this discussions with colleagues and advisors:

- **3D Player Pose Estimation:** A natural next step is to extend our vision capabilities from 2D segmentation to 3D understanding of players. Estimating full 3D poses of players during gameplay would provide rich information about player actions and conditions (e.g., a player's skating stride, or whether a player is crouching to block a shot). This is extremely challenging from a single broadcast video, but a combination of multi-view footage and simulation could help. Future work could leverage our Unity tool to generate synthetic data of players in various poses, or integrate multi-camera calibration to estimate 3D pose from real games. Success in 3D pose estimation would enable analyses of player biomechanics and possibly injury risk, as well as more fine-grained action recognition.
- **Enhanced Homography and Camera Tracking:** This pipeline for aligning video to the rink relies on either manual annotation for static cameras or a learning-based homography regressor for dynamic cameras. One future direction is to incorporate real-time camera pose estimation, determining the 3D position and orientation of the

broadcast camera for each frame. This could be achieved via detecting rink markings and solving for camera extrinsics, or using SLAM techniques.

- **Robust Multi-Object Player Tracking:** There is potential for significant improvements in tracking and identification. This includes integrating state-of-the-art tracking-by-detection methods or GNN, as well as jersey number or facial recognition for ID labeling. Simulation data can support OCR model training for jersey numbers. Future work could also expand to puck tracking, which might be tackled via simulation-assisted training of small-object detectors.
- **Realistic Motion Synthesis in Simulation:** Currently, the system uses interpolated or manually scripted paths. Future enhancements could include deep learning models that generate plausible trajectories based on game context, or physics-based animation to simulate skating and collisions. This would improve realism and broaden use cases such as scenario testing and physical interaction modeling (e.g., checking).
- **Improving Unity Rendering and Assets:** To close the realism gap, efforts could focus on better 3D player models, enhanced lighting and ice reflection, dynamic rink designs, jersey variations, and Unity HDRP rendering. These would help reduce domain shift and increase the fidelity of synthetic frames used for training.
- **Automated Action Recognition:** A high-level goal is the recognition of complex gameplay events (e.g., body checks, set plays). This simulation tool can render labeled scenarios to train such models. Future efforts might involve multi-view video and combining pose, stick, and puck data to classify actions.
- **Unified Augmentation-Simulation Pipeline:** A longer-term vision involves combining SLDA and simulation into a closed-loop system. For example, an augmented model improves detection, leading to better simulation inputs, which generate new training data, closing the feedback loop. This would require tight integration and possibly active learning strategies, but promises a robust, adaptive analytics pipeline.

In conclusion, the future work spans from low-level improvements (better models, better graphics) to high-level integration (holistic systems combining everything). The common thread is pushing towards more comprehensive and intelligent hockey analytics: from identifying a tiny blurred puck on the ice to understanding a full strategic play involving all players. The work done in this thesis provides a stepping stone in that direction, and we are optimistic that building upon these contributions will eventually lead to truly smart sports analytics systems that can see and understand the game as well as a seasoned human analyst, if not better.

References

[1] A. Aakanksha, S. Yarlagadda, and S. Kaul. Improving robustness of semantic segmentation to motion-blur using class-centric augmentation. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 13426–13435, 2023.

[2] Hassan Abu Alhaija, Siva Karthik Mustikovela, Lars Mescheder, Andreas Geiger, and Carsten Rother. Augmented reality meets computer vision: Efficient data generation for urban driving scenes. *International Journal of Computer Vision*, 2018.

[3] M. Andriluka, U. Iqbal, A. Milan, E. Insafutdinov, L. Pishchulin, J. Gall, and B. Schiele. Posetrack: A benchmark for human pose estimation and tracking. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[4] B. Balaji and D. A. Clausi. Towards agile human pose estimation: A case study in ice hockey analytics. *Journal of Computational Vision and Imaging Systems*, 9(1):27–33, 2023.

[5] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. Yolov4: Optimal speed and accuracy of object detection, 2020.

[6] S. Borkman et al. Unity perception: Generate synthetic data for computer vision, 2021. arXiv:2107.04259.

[7] G. Brasó and L. Leal-Taixé. Learning a neural solver for multiple object tracking, 2020. arXiv preprint.

[8] Lewis Bridgeman, Marco Volino, Jean-Yves Guillemaut, and Adrian Hilton. Multi-person 3d pose estimation and tracking in sports. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2019.

[9] Yohann Cabon, Naila Murray, and Martin Humenberger. Virtual kitti 2. *arXiv preprint arXiv:2001.10773*, 2020.

[10] Yizheng Cai, Nando Freitas, and J. J. Little. Robust visual tracking for multiple targets. In *Computer Vision – ECCV 2006*, pages 107–118. 2006.

[11] Zixi Cai, Helmut Neher, Kanav Vats, David A. Clausi, and John Zelek. Temporal hockey action recognition via pose and optical flows. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2019.

[12] Z. Cao, T. Simon, S.-E. Wei, and Y. Sheikh. Realtime multi-person 2d pose estimation using part affinity fields. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 7291–7299, 2017.

[13] B. Chen and J. J. Little. Sports camera calibration via synthetic data. In *Proceedings of the ICCV Workshops*, 2019.

[14] A. Cioppa, S. Giancola, A. Deliege, L. Kang, X. Zhou, Z. Cheng, B. Ghanem, and M. Van Droogenbroeck. Soccernet-tracking: Multiple object tracking dataset and benchmark in soccer videos, 2022. Accepted to CVSports Workshop at CVPR 2022. arXiv:2204.06918.

[15] Code This Lab. Hockey player animated, 2021. Unity Asset Store. Version 1.0, Released July 4, 2021.

[16] Yutao Cui, Chenkai Zeng, Xiaoyu Zhao, Yichun Yang, Gangshan Wu, and Limin Wang. Sportsmot: A large multi-object tracking dataset in multiple sports scenes. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2023.

[17] J. Deng, W. Dong, R. Socher, L. J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2009.

[18] Jiangfan Deng, Dewen Fan, Xiaosong Qiu, and Feng Zhou. Improving crowded object detection via copy-paste. In *AAAI Conference on Artificial Intelligence (AAAI)*, pages 497–505, 2023.

[19] Maximilian Denninger, Martin Sundermeyer, Dominik Winkelbauer, Youssef Zidan, Dmitry Olefir, Mohamad Elbadrawy, Ahsan Lodhi, and Harinandan Katam. Blenderproc, 2019.

[20] Nikita Dvornik, Julien Mairal, and Cordelia Schmid. Modeling visual context is key to augmenting object detection datasets. In *European Conference on Computer Vision (ECCV)*, pages 364–380. Springer, 2018.

[21] D. Dwibedi, I. Misra, and M. Hebert. Cut, paste and learn: Surprisingly easy synthesis for instance detection. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2017.

[22] H. Fang et al. Instaboost: Boosting instance segmentation via probability map guided copy-pasting. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2019.

[23] A. Gaidon, Q. Wang, Y. Cabon, and E. Vig. Virtual worlds as proxy for multi-object tracking analysis. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4340–4349, 2016.

[24] B. Ghiasi et al. Simple copy-paste is a strong data augmentation method for instance segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021.

[25] Google Research. Google research football, 2019. GitHub repository.

[26] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 2980–2988, 2017.

[27] Q. He, Y. Duan, Z. Yang, Y. Wang, L. Yang, L. Bai, and L. Zhao. Context-aware augmentation for liver lesion segmentation: shape uniformity, expansion limit and fusion strategy. *Quantitative Imaging in Medicine and Surgery*, 13(8):5043–5057, 2023.

[28] Dan Hendrycks and Thomas Dietterich. Benchmarking neural network robustness to common corruptions and perturbations, 2019.

[29] Mehdi Houshmand Sarkhoosh, Sushant Gautam, Cise Midoglu, Saeed Shafiee Sabet, Tomas Kupka, and Pål Halvorsen. Hockeyrink: A dataset for precise ice hockey rink keypoint mapping and analytics. In *Proceedings of the 16th ACM Multimedia Systems Conference (MMSys '25)*, pages 249–255, 2025.

[30] Arthur Juliani, Vincent-Pierre Berges, Ervin Teng, Andrew Cohen, Jonathan Harper, Chris Elion, Chris Goy, Yuan Gao, Hunter Henry, Marwan Mattar, and Danny Lange. Unity: A general platform for intelligent agents. *arXiv preprint arXiv:1809.02627*, 2018.

[31] Victor Khaustov and Maxim Mozgovoy. Recognizing events in spatiotemporal soccer data. *Applied Sciences*, 10(22):8046, 2020.

[32] J. T. Kim, T. Chen, Z. Dong, and N. Kunchala. Centralized copy-paste: Enhanced data augmentation strategy for wildland fire semantic segmentation, Nov 2023. arXiv:2311.03980.

[33] M. Kurach et al. Google research football: A novel reinforcement learning environment, 2019. arXiv:1907.11180.

[34] T.-Y. Lin, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, C. L. Zitnick, and P. Dollár. Microsoft coco: Common objects in context. In *European Conference on Computer Vision (ECCV)*. arXiv, 2014.

[35] Hao Liu, Song Bai, Shiguang Shan, and Tieniu Huang. Pose-guided r-cnn for jersey number recognition in sports. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2019.

[36] S. Liu, Y. Zhang, M. Shah, and M. Chang. Simpose: A synthetic dataset for human pose estimation in sports, 2021. arXiv:2104.09486.

[37] David A. Clausi John Zelek Mehrnaz Fani, Pascale Berunelle Walters and Alexander Wong. Localization of ice-rink for broadcast hockey videos, 2021. arXiv:2104.10847.

[38] K. Okuma, A. Taleghani, D. Freitas, J. Little, and D. Lowe. A boosted particle filter: Multitarget detection and tracking. In *Proceedings of the European Conference on Computer Vision (ECCV)*, volume 3021, 2004.

[39] Patrick Pérez, Michel Gangnet, and Andrew Blake. Poisson image editing. *ACM Transactions on Graphics (SIGGRAPH)*, 22(3):313–318, 2003.

[40] H. Prakash. Broadcast is all you need: Robust multiplayer tracking in ice hockey using monocular videos, 2025.

[41] H. Prakash et al. Multi player tracking in ice hockey with homographic projections. In *CVPR Workshop*, 2024.

[42] Weichao Qiu and Alan Yuille. Unrealcv: Connecting computer vision to unreal engine. *arXiv preprint arXiv:1609.01326*, 2016.

[43] Erik Reinhard, Michael Ashikhmin, Bruce Gooch, and Peter Shirley. Color transfer between images. *IEEE Computer Graphics and Applications*, 21(5):34–41, 2001.

[44] Stephan R. Richter, Vibhav Vineet, Stefan Roth, and Vladlen Koltun. Playing for data: Ground truth from computer games. In *European Conference on Computer Vision (ECCV)*, pages 102–118. Springer, 2016.

[45] German Ros, Laura Sellart, Joanna Materzynska, David Vazquez, and Antonio M. Lopez. The synthia dataset: A large collection of synthetic images for semantic segmentation of urban scenes. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3234–3243, 2016.

[46] L. Salass et al. Ice hockey puck localization using contextual cues, Jun 2025. arXiv:2506.04365v1.

[47] Mehdi Houshmand Sarkhoosh, Sushant Gautam, Cise Midoglu, Saeed Shafiee Sabet, Tomas Kupka, and Pål Halvorsen. Hockeyorient: A dataset for ice hockey player orientation classification. In *Proceedings of the 16th ACM Multimedia Systems Conference (MMSys '25)*, pages 277–283, New York, NY, USA, 2025. Association for Computing Machinery.

[48] Manolis Savva, Abhishek Kadian, Oleksandr Maksymets, Yili Zhao, Erik Wijmans, Bhavana Jain, Julian Straub, Jia Liu, Vladlen Koltun, Jitendra Malik, Devi Parikh, and Dhruv Batra. Habitat: A platform for embodied ai research. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019.

[49] Shital Shah, Debadeepta Dey, Chris Lovett, and Ashish Kapoor. Airsim: High-fidelity visual and physical simulation for autonomous vehicles. *arXiv preprint arXiv:1705.05065*, 2017.

[50] C. Shorten and T. M. Khoshgoftaar. A survey on image data augmentation for deep learning. *Journal of Big Data*, 6:60, 2019.

[51] Bharat Singh and Larry S. Davis. An analysis of scale invariance in object detection: Snip. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3578–3587, 2018.

[52] Hao Su, Charles R. Qi, Yangyan Li, and Leonidas J. Guibas. Render for cnn: Viewpoint estimation in images using cnns trained with rendered 3d model views. In *IEEE International Conference on Computer Vision (ICCV)*, 2015.

[53] Jonas Theiner and Ralph Ewerth. Tvcalib: Camera calibration for sports field registration in soccer. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, pages 1166–1175, 2023.

[54] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 23–30, 2017.

[55] Jonathan Tremblay, Thang To, and Stan Birchfield. Falling things: A synthetic dataset for 3d object detection and pose estimation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, June 2018.

[56] Tronitec Game Studios. Ice hockey arena, 2015. Unity Asset Store. Version 1.1, Released December 15, 2015.

[57] Yi-Hsuan Tsai, Xiaohui Shen, Zhe Lin, Kalyan Sunkavalli, Xin Lu, and Ming-Hsuan Yang. Deep image harmonization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[58] Gül Varol, Javier Romero, Xavier Martin, Naureen Mahmood, Michael J. Black, Ivan Laptev, and Cordelia Schmid. Learning from synthetic humans. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 109–117, 2017.

[59] K. Vats, M. Fani, D. A. Clausi, and J. S. Zelek. Evaluating deep tracking models for player tracking in broadcast ice hockey video, 2022. Unpublished.

[60] Kanav Vats, Mehrnaz Fani, David A. Clausi, and John S. Zelek. Ice hockey player identification via transformers and weakly supervised learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2022.

[61] Kanav Vats, William McNally, Chris Dulhanty, Zhong Qiu Lin, David A. Clausi, and John Zelek. Pucknet: Estimating hockey puck location from broadcast video. *arXiv preprint arXiv:1912.05107*, 2019.

[62] Ning Xu, Brian Price, Scott Cohen, and Thomas Huang. Deep image matting. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[63] J. Zhang, Y. Zhang, and X. Xu. Objectaug: Object-level data augmentation for semantic image segmentation, 2021. arXiv:2102.00221.

[64] Lingzhi Zhang, Tarmily Wen, Jie Min, Jiancong Wang, David Han, and Jianbo Shi. Learning object placement by inpainting for compositional data augmentation. In *European Conference on Computer Vision (ECCV)*. Springer, 2020.

Glossary

Copy-Paste An object-level data augmentation method where object masks are cut from one image and pasted into another 14, 30, 31, 37, 39–42, 44, 45

HTD-1 The first version of the Hockey Tracking Dataset, a proprietary dataset containing player tracking annotations in broadcast footage 2, 9, 10

HTD-2 An open-source version of the Hockey Tracking Dataset with annotated player bounding boxes and IDs from NHL games 2, 9, 10, 28

InstaBoost A context-aware data augmentation method that places objects based on learned likelihood maps and applies jittering for realism 14, 17, 26, 28

instance-level data augmentation Augmentation that involves inserting or modifying specific object instances within an image, rather than transforming the whole image 3, 5

MOT Multi-Object Tracking, the task of simultaneously tracking multiple objects across frames while maintaining consistent identities 19, 28

ObjectAug An object-level data augmentation method proposed for semantic image segmentation. It augments training data by inserting object instances into different contexts, improving model generalization and robustness 30, 31, 37, 45

PLUCC Puck Localization Using Contextual Cues; a model that leverages player positions and orientations to infer puck location 21, 28

SoccER dataset A large-scale synthetic dataset for soccer event recognition, consisting of simulated gameplay with ground-truth event annotations 11

Synthetic Local Data Augmentation An instance-level data augmentation method that synthetically inserts realistic hockey stick instances into images 4, 5, 15

Unity-based 3D Hockey Simulation Tool A custom simulation platform developed in this thesis that reconstructs hockey games in 3D from structured game data (e.g., player/puck coordinates or trajectories). It allows flexible camera control, multi-view rendering, and synthetic video generation, providing a framework for analysis, visualization, and data creation in hockey research iv, 4, 7, 24, 77, 78, 81, 83, 86, 87, 90

Virtual KITTI A synthetic clone of the KITTI dataset, generated using a game engine to evaluate vision models in driving scenarios 11, 25, 26, 28