Estimating the Impact of Automated Umpiring in Baseball via Monte Carlo Simulation

by

Keithen Shepard

B.S. Electrical Engineering and Computer Science, Massachusetts Institute of Technology 2021

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 2022

Author .....
Department of Electrical Engineering and Computer Science
May 06, 2022

Certified by .....
Anette Peko Hosoi
Neil and Jane Pappalardo Professor, Mechanical Engineering
Thesis Supervisor

Accepted by .....
Katrina LaCurts
Chair, Master of Engineering Thesis Committee

Estimating the Impact of Automated Umpiring in Baseball via Monte Carlo Simulation

by

Keithen Shepard

Submitted to the Department of Electrical Engineering and Computer Science
on May 06, 2022, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

The MLB (Major League Baseball) has made multiple changes to the game of baseball recently to enhance the viewing experience for fans. One viable idea that has been tossed around for multiple years has been the implementation of an automated umpiring system. The MLB has the technology to utilize such a system using Trackman technology however most MLB teams have expressed opposition to the idea. Using an automated system would get rid of human mistakes that umpires make due to the high-speeds of MLB pitches and other challenges.

We present a method to estimate the impact of automated umpiring given MLB pitch data. We define a novel pipeline for simulating the statistical changes in MLB games following the correction of umpire mistakes. This pipeline uses historical game data to guide our estimations and then compares our findings to the baseline real game statistics. We finally use this pipeline to analyze the changes that an automated umping model would bring on average to the MLB game.

Thesis Supervisor: Anette Peko Hosoi
Title: Neil and Jane Pappalardo Professor, Mechanical Engineering

Acknowledgments

I would first like to start out by thanking Peko for all of her support throughout the whole MEng process. Ever since our first meeting she has been nothing but eager to help with any questions I've had and this work would not have been possible without her guidance. I also would like to say a big thank you and I love you to my parents, little brothers, and my whole family who have helped me get to this point. It would be impossible to put into words how thankful I am for everything you all have done for me to help me grow as student, athlete, and a person. Thank you to all of my fellow brothers, both young and old, of Delta Kappa Epsilon, especially the 9 of you that are a part of my pledge class. Although we may have only gotten 2.5 years living in the house due to COVID, the memories that I've made with you all are some of my most cherished and I wouldn't trade them for anything. Deciding to complete this MEng is due in part to the amazing group of people I was surrounded with at DKE and their encouragement and I am thankful to have learned from the best. Special shout out to my girlfriend who has helped me grow and brought so much happiness into my life since we met my sophomore year. Your never ending support has kept me focused on the big picture even as some days got very stressful. Thank you as well to Mr. Dan West, the best chef/boomer around. You have done everything and more in terms of feeding and helping our DKE class to enjoy our final years of college. One of my best experiences of MIT has been playing football for MIT's football team. A huge thank you to Coach Bubna, Perron, Brennan, and all the other coaches and players for making one of the hardest places in the world still one of the most fun. With all of the obligations students have at MIT, I am so grateful to have been able to compete with the football team day in and day out to try and dominate the world. As I submit this thesis for consideration, I just want to show appreciation for all those who have played a part in my life, small or large, as everything I have learned and experienced has ultimately helped me get to this point. love you Mom, Dad, Cason and Reid, Roll Tech, and kerothen philoi aei.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

Baseball has long been known as "America's past time" ever since its introduction to the U.S. in 1846 [5]. As the game progressively grew initially in reach and popularity, so too did the need for baseball technology to advance in order to meet the needs of the game. Often, when technology is discussed in the field of sports, the focus is on creating analyses that will improve the performance of the players such that the quality of the sport increases. Teams nowadays will spend upwards of millions of dollars, with a recent source claiming the Dodgers, one of the teams in the MLB, spends upwards of $20 million dollars annually on baseball technology and research [22]. Given the broad range of data points that can be extrapolated from a baseball game, the number of data-based decisions that a team can make is vast. Many executives of the MLB are currently looking towards data analysis not only as a means of benefiting the performance of the players but also as a way to identify changes that will help reinvigorate the fan bases that give meaning to the sport.

Despite such a rich history and long standing fan bases, the MLB is facing a problem that other professional sports leagues in the U.S. have not had to deal with: declining viewership. During each regular season, all 30 MLB teams play a total of 162 games each totaling $\frac{162 \cdot 30}{2} = 2430$ games for one season. Most casual fans do not have the time to attend the longer games during the week and still make it to work the

next morning. The number of fans that attended games in 2007 was approximately 79 million, while attendance only reached about 68 million in 2019, a decrease of almost 14% for in person viewership [2]. On average, as of 2007, the average MLB baseball game lasts around 2 hours and 50 minutes and hasn't dropped below this duration since. In comparison, a typical game lasted 2 hours and 30 minutes in the 1970s [21].

Given these statistics, MLB executives have been looking for different ways to make games more exciting and appealing to the everyday fan. One such change to baseball games that has been proposed to make in-person viewing more appealing to fans is to automate umpiring [4]. The MLB currently has the technology to detect whether or not every single pitch against a given batter should have been called a strike. Although this technology exists, the MLB still relies on human umpires to make these ball-strike decisions throughout games. This thesis work focuses not on the implementation of these umpire-replacing technologies, but on analyzing the changes that we might see in MLB games if automated systems were used to call balls and strikes. Specifically, we will look at how some key statistics that are closely related to fan engagement change: number of home runs hit, number of balls hit in play, and number of pitches in game.

1.2 Contributions

Here we introduce some of the key contributions presented in this thesis. These contributions are generated using MLB pitching data, which we will describe in detail in section 3.2.

1.2.1 Quantifying Umpire Mistakes

Using the idea of a static strike zone predefined by a batter's size, we create a function to evaluate pitches whose outcomes are currently decided by an umpire. By using a bounding box on the strike zone and the ball's tracking location as it crosses home plate, we are able to simulate an automated umpire model that classifies pitches as correctly or incorrectly called. Classifying these mistakes gives us the ability to

counter-factually correct the mistake and simulate future events in the game given the new starting state. We will also analyze different patterns in umpire mistakes.

1.2.2 Extracting Baseball Game State Transition Likelihoods

In order to accurately model the sequence of events that can occur in a baseball game, we define a baseball game state with several features that define all the relevant information about our game at a given point in time. We will discuss the state space size and how it is conducive to producing results that are statistically significant. We then perform aggregations over our MLB dataset to generate a probabilistic transition model, otherwise known as a Markov Chain, for moving from one state to another state through an event such as a pitch. This Markov Chain is then used as a basis for game simulation in our Monte Carlo approach.

1.2.3 Counterfactual Monte Carlo Simulation Evaluation

After generating our transition matrices and classifying mistakes, we will describe a Monte Carlo method for extrapolating through innings in which mistakes occurred to look at statistical differences between the actual statistics and expected statistics given an automated umpiring system. We will focus on trends in differences between statistics on an aggregate level as well as on a per inning level. By comparing means and standard deviations of expected values we can identify areas where MLB games could change either for the benefit or harm of viewership as defined previously by utilizing measured ball trajectories.

1.3 Outline

In Chapter 2, we will discuss work that has been conducted in the area of baseball technology and umpire strike zone analysis. In Chapter 3, we will define the dataset we are using for analysis as well as look at a method to classify umpire mistakes using this dataset. In Chapter 4, we will explain our game state representation and

methodology for computing transition matrices both between states and for state-event pairs. In Chapter 5, we present a novel approach to simulating game statistics while utilizing our classified mistakes and transition matrices. In Chapter 6, we present evaluation metrics used to both validate simulation findings and to compare findings with actual game results. Chapter 7 presents some possible future extensions to build upon this research as well as discussion about key insights this work provides regarding what effects implementing an automated umpiring system would have on the MLB game.

Chapter 2

Related Work

This section will discuss the background work done so far in this paper's area of study. To the best of our knowledge, there are not any direct studies that look at the cause and effect relationship of an automated umpiring model on the statistical results of MLB games, however there are relevant studies that discuss separate pieces of this thesis pipeline.

2.1 Baseball Technology

With the rise of technology in our everyday lives, so to has technology spread rapidly in the field of sports. Baseball especially has been further expanding and improving data collection efforts through technological improvements. Most specifically, these efforts are being led by Trackman, a baseball tracking system that all professional teams in the MLB utilize to analyze everything from pitchers to batters [10]. Notably, in the job space of umpiring, an interesting study by Brian Mills looked into how the development of tracking and evaluation technology in the MLB actually helped improve the quality of umpiring in MLB games [19]. This study explains how with the advancement of QuesTec and later on the Zone Evaluation system (two systems for monitoring and evaluating pitch calls made by umpires) allowed umpires to improve in their skill at a rate consistent with the technological improvement. This suggests the possibility of growth in the game of baseball with the advancement of an automated

umpiring system.

2.2 Umpire Consistency & Shape of the Strike Zone

One large simplification we use in this study is that the strike zone is a dynamically heightened rectangle of static width. This is actually not the case in current MLB games. Many studies look at the effect that the pitch count or the handedness of the batter can have on what borderline pitches are called by umpires [16]. One specific study analyzes potential reasons against using a robot umpire. One reason was that umpires try and err on the side of not deciding an at bat [12]. This means that if a batter is ahead in the count, i.e. they have more balls than strikes (3-1 count or 2-0 count), the umpire will call a more relaxed strike zone in order to try and avoid deciding the at-bat and walking the batter on a called ball. The same goes for when a batter is behind in the count; the strike zone contracts and the umpire calls more balls. Other studies examine the effect that different individual umpires have on the kind of calls made throughout the game and how each of their respective strike zones differ [11]. Many studies also agree that the strike zone is much better represented by an oval shape that expands and contracts both horizontally and vertically as opposed to a rectangle [13]. All of these studies' findings lie outside the scope of this thesis as we are concerned not with the technology or models that define how we could automate the strike zone, but once we have an accurate automated zone what are the effects on statistics throughout the game.

Chapter 3

Classifying Umpire Mistakes from MLB Pitching Data

Now that we have laid the background for this problem and discussed some of the previous work in these areas, we will look at the first step in performing our analysis: classifying pitches as correctly or incorrectly called based on baseball tracking data. This section will first give some high level background on the game of baseball and some terminology we will use throughout the rest of the thesis. We will then move on to discussing the MLB dataset we are using for our analysis before discussing the criterion for classifying pitches.

3.1 Baseball Background

Before we dive into the dataset we will be using, first we will define a few of the key baseball terms that will be critical to understanding our work. This is by no means a comprehensive explanation of the rules or details of baseball and serves only as a guide for the immediate knowledge needed to understand our algorithms. Some rules of the game may be oversimplified in order to emphasize aspects of the game that are essential to understand.

Every baseball game consists of at least 9 innings, each with a top and bottom half. During each half inning, one team is the **batting** team and one team is the

fielding team. The fielding team has 9 players on the field with one being the **pitcher** who tries to throw the baseball so the hitter cannot hit it. The batting team has one hitter up at a time who stands at home plate and tries to hit the ball before advancing around all 4 bases to score a run. The team with the most runs scored at the end of 9 innings wins. In the event of a tie, extra innings are played until one team is ahead after the bottom half of the inning.

Every pitch is called a ball or a strike: a ball is a pitch that crosses home plate outside of the strike zone while a strike is a pitch that crosses home plate inside of the strike zone. These decisions on whether or not a pitch is in the strike zone are determine by an umpire who watches from behind home plate. The umpire makes a pitch call decision only if the batter does not swing at the ball. For any at bat, 3 strikes means the hitter is out while 4 balls means the hitter may go advance to 1st base. The number of strikes and ball for each at bat are kept track of in a count, for example an at bat with 2 balls and 1 strike so far is a $2-1$ count. In order to end a half inning and get to bat, the fielding team must get 3 outs, either by striking out the batters, catching a hit ball in the air, or fielding a ball on the ground and throwing it to a base before the runner arrives. Figure 3-1 shows the layout of a baseball field.

The key statistics we will be keeping track of throughout our analysis are total number of pitches thrown, home runs hit, and hits. A **hit** is an event in which a batter makes contact with the ball and is able to safely make it to one of the bases. There are different types of hits that we will reference: a **single** is a hit where the batter advances to 1st base, on a **double** the runner advances to 2nd base, on a **triple** the runner advances to 3rd base, and on a **home run** the runner advances all the way to home plate and scores a run. A home run is any hit by a batter that goes over the fence of the stadium inside the foul lines.

3.2 MLB Pitching Dataset Description

This thesis work is formulated using a dataset created in April 2020 with MLB pitching data from the 2017-2020 seasons. Each row of the table corresponds to one pitch

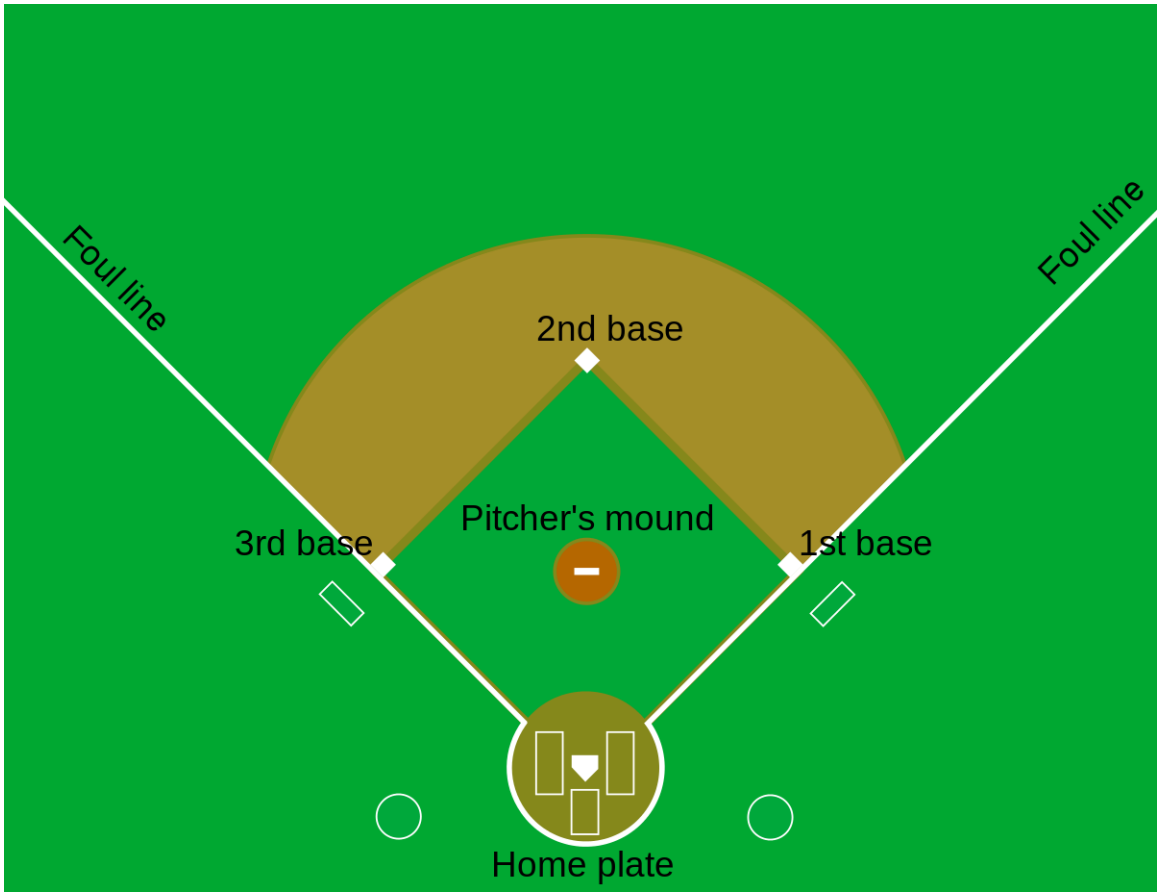

Figure 3-1: An overhead view of a baseball field with the pitchers mound and 4 bases labeled.

in a game during one of these seasons. Table 3.1 displays the number of unique games and pitches in our dataset.

For each pitch, metadata information about the game is available such as which teams are playing, which player is pitching or hitting, where runners are on base, what temperature it was outside, and even what stadium the game is being played in. Aside from the metadata, each pitch also holds specific tracking data specifying where and how the ball traveled when it was thrown (and possibly hit). This includes fields such as velocity, spin rate, vertical location as it crosses home plate, horizontal

Number of Unique Games	7,276
Number of Unique Pitches	2,168,790

Table 3.1: Key statistics from the MLB pitches dataset.

location compared to the middle of home plate, exit angle, and many other pieces of location tracking information. Finally, each row also holds information about the results after a pitch. This includes information such as whether or not the batter swung, what the pitch call was, what the resulting count is, how many outs there are, and whether or not the hitter successfully hit the ball.

While a subset of the data fields in this dataset are publicly available on places such as Kaggle [4], this dataset has much more detailed and thorough features that we can use to help define our in game situation before and after any given pitch. For the sake of umpire mistake classification, we will next define what exactly a strike zone is and how the home plate tracking data for a pitch can be mapped onto a given strike zone.

3.3 Mistake Classification

Initially, we will focus on the fields of our dataset we discussed in 3.2 that allow us to decide whether any given pitch call p_i should be classified as a mistake or not. There are two pieces to determining this classification: the pitch location and the location of the strike zone. This seems trivial, as when watching on television this box often appears as a stationary static rectangle. The strike zone is in fact static in its width, as it is always equal to the width of home plate which is 17 inches. However, the strike zone height changes according to the size of the batter up to bat. The umpire is in charge of deciding the extent of this strike zone at the moment the batter starts their swing [20].

Per the MLB's official definition of a strike zone, it is defined as the

> *"midpoint between a batter's shoulders and the top of the uniform pants, when the batter is in his stance and prepared to swing at a pitched ball, and a point just below the kneecap..."* [9]

Given this definition, we can see the height of the strike zone is a function of the batters height, as batters who are taller will tend to have a larger distance from their

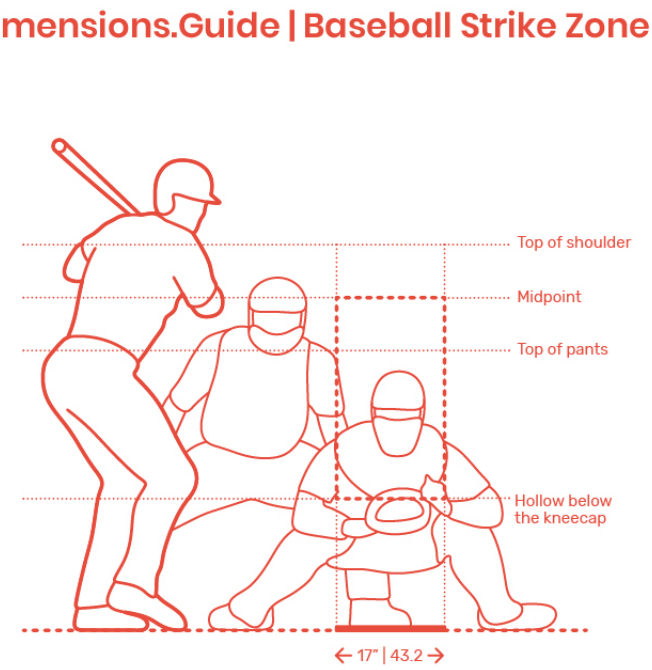

Figure 3-2: Formulation of a strike zone per the MLB's definition [1]

.

shoulders to the ground, raising the top of the strike zone higher. Figure 3-2 allows us to visualize how this zone is calculated by the umpire from a given batter. Rather than formulating our own method for determining these upper and lower bounds on the zone, we rely on the MLB's technology and dataset. To account for this, our dataset has a pair of fields that allow us to define the strike zone for any given at bat based on the batter's size.

- `SZ_Top` - the height in feet of the top of the strike zone
- `SZ_Bottom` - the height in feet of the bottom of the strike zone

For our analysis, we will define the following correlations between Figure 3-2 and our data fields: `SZ_Top` = "Midpoint" and `SZ_Bottom` = "Hollow below the kneecap".

Now that we have defined how a strike zone is calculated by an umpire, we can dive into actually classifying any pitch as correctly or incorrectly called given our

definitions. In order to accurately classify a pitch, we must know the exact location where our pitch crosses home plate and what it was called as. For this, we have three data points for every pitch we will look at:

- `PlateLocSide` - the distance from the center of Home Plate in feet
- `PlateLocHeight` - the distance from the ground in feet
- `PitchCall` - the outcome of the pitch $\in$ {Ball, Called Strike}

In order to more clearly define our framework for tracking pitch location, Figure 3-3 depicts an example strike zone with a given pitch and how these metrics are interpreted. With reference to baseball terminology, we can think of classifying a

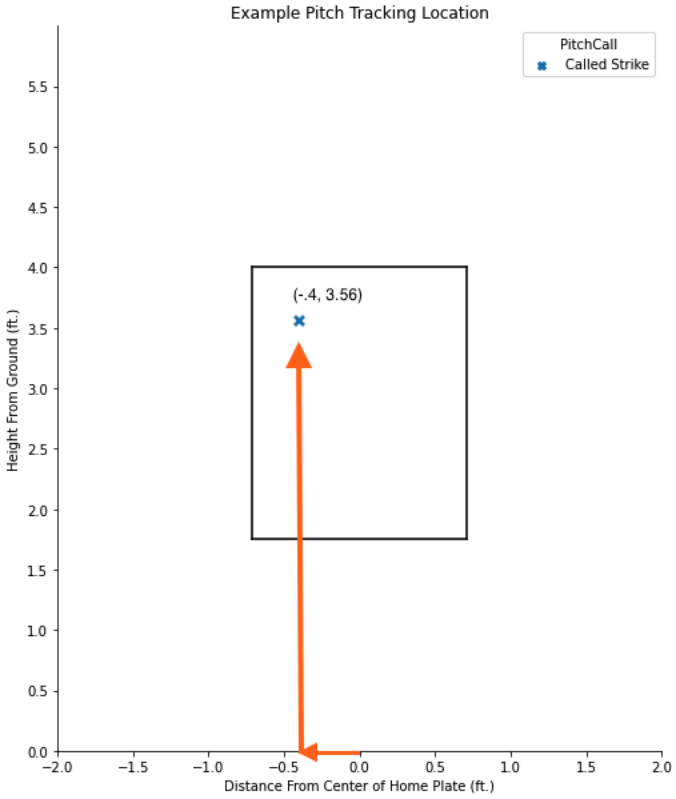

Figure 3-3: Visual representation of an example strike zone and a pitch's tracking data crossing home plate at location `(PlateLocSide, PlateLocHeight)`.

pitch as a mistake if it was called a strike but is outside the strike zone or if it was called a ball and was inside the strike zone. Using both our definition of the strike zone

in conjunction with the pitch location information, we define some boolean checks for whether a pitch falls within different portions of our strike zone.

- $\texttt{InZoneWidth} = \texttt{PlateLocSide} \geq \frac{-8.5}{12}$ and $\texttt{PlateLocSide} \leq \frac{8.5}{12}$
- $\texttt{InZoneHeight} = \texttt{PlateLocHeight} \geq \texttt{SZ\_Bottom}$ and $\texttt{PlateLocHeight} \leq \texttt{SZ\_Top}$

Using the predefined checks for a given pitch, we can now define the following formal method for classifying a pitch call p_i as a mistake or not, with 1 being the *mistake* classification and 0 being the *correct* classification.

$$f(p_i) = \begin{cases} 1, \text{ if } \big(\texttt{InZoneWidth} \wedge \texttt{InZoneHeight}\big) \wedge p_i \in \{Ball\} \\ 1, \text{ if } \big(\neg\texttt{InZoneWidth} \vee \neg\texttt{InZoneHeight}\big) \wedge p_i \in \{CalledStrike\} \\ 0, \text{ else} \end{cases} \tag{3.1}$$

Using this method to classify our pitches dataset, we can now look at how mistakes umpires make trend with the type of call they are making. Table 3.3 shows metrics on all pitches in our dataset that an umpire must make a decision on. As we can see, approximately 12% of the time, an umpire makes a mistake. On average, the number of pitches an umpire must make a call on is ≈ 150. Given our accuracy rate of 87.823%, this means there are an average of ≈ 19 pitches in a game that are mistakes. Looking at the holistic result of one game can give us a better picture of what these mistakes tend to look like. Figure 3-4 charts all 121 pitches in a game between the Phillies and Mets in 2017 where the umpire had to decide between Ball or Strike. Of those 121 pitches, 47 were called strikes while 16 of those were mistakes (34% mistake rate), while 74 were called balls and only 3 were mistakes (4% mistake rate). The static strike zone visualized for this figure is the result of averaging all strike zone upper and lower boundaries over our dataset to give us the closest estimate. For this reason, there may be some pitches correctly classified as mistakes that appear to not be mistakes according to this zone.

Pitch Call	**Correct**	**Mistakes**	**# of Pitches**	**% Mistake**
Called Strike	243,434	118,289	361,723	32.702
Ball	715,958	14,736	730,694	2.017
{Called Strike $\wedge$ Ball}	959,392	133,025	1,092,417	12.177

Table 3.2: Metrics on the classifications of MLB pitches as mistakes.

As we can see, many of the mistakes have a small margin of error with the definition of the strike zone. In a game as competitive as baseball where less than an inch can decide a pitch call, these mistakes can hold large consequences. Depending on the game situation when these mistakes occur, those pitch calls can drastically alter both the outcome of the at-bat as well as the rest of the inning. In order to analyze exactly how correcting these mistakes could have changed the statistics in the game following the pitch, we must first look at the probabilistic transitions that occur from one game state to another.

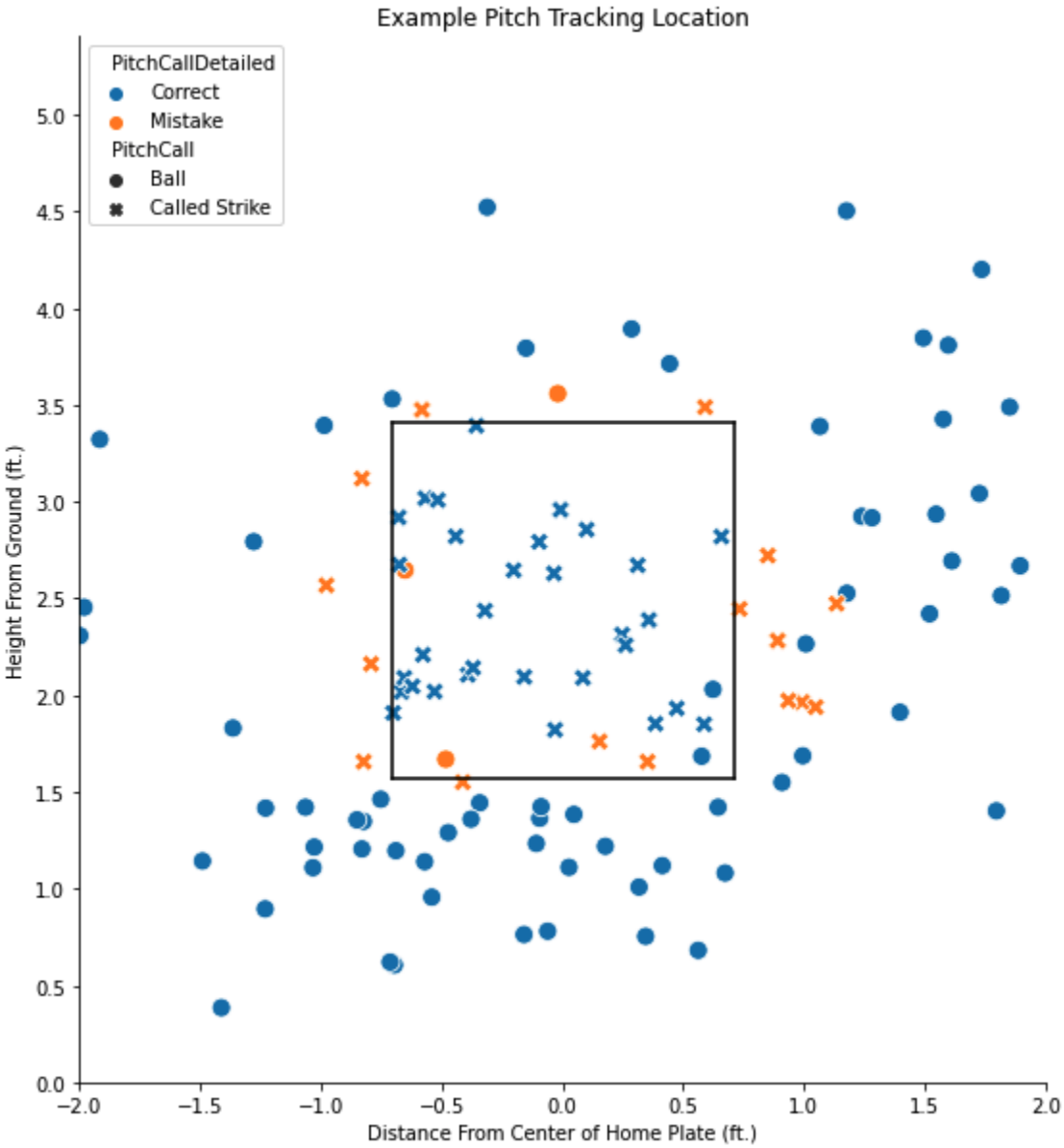

Figure 3-4: All charted pitches decided by an umpire in a game between the Phillies and Mets on 04/17/2019.

Chapter 4

Extracting Baseball Game State Transition Matrices

Once we have our pitches dataset classified according to an automated umpiring model, we now must look to features about the game state in order in order to generate transition matrices that can approximate the actual progression of a baseball game. Once we discuss how we define the state of a game before any pitch, we will move on to talk about how we generate two different transitional models that we can use to simulate events in a game: a state to state Markov Chain and a state-event probability table.

4.1 Game State Representation

In order to define our game state representation, we first have to think about the size of our state space. State representation learning is a form of feature learning in which an agent aims to learn about a domain of data by decomposing it into features from which it can characterize the domain [18]. Baseball is similar in the fact that given a handful of features that define the state of the game, one can learn what each feature means and characterize how the game may progress forward. Rather than learning the relationship of these features through intentionally choosing actions, we will instead rely on actual baseball game data to create aggregate probabilistic transitions between

states and in turn their features.

First, we must define what our state representation will be such that we will have statistically significant probability distributions over possible transitions when we create our matrices. The features we look at are **Inning**, **Outs**, **Runners on Base**, **Runners in Scoring Position**, **Balls**, and **Strikes**. All 6 of these fields are available for every pitch in our MLB dataset. The following defines how we bin some of the values together to create a state vector s_i:

- Innings:= $I_i = \begin{cases} 0, \text{ if Inning} = \text{1-3} \\ 1, \text{ if Inning} = \text{4-6} \\ 2, \text{ if Inning} = 7+ \end{cases}$
- Outs:= $O_i = \begin{cases} 0, \text{ if Outs} = 0 \\ 1, \text{ if Outs} = 1 \\ 2, \text{ if Outs} = 2 \end{cases}$
- Runners on Base:= $ROB_i = \begin{cases} 0, \text{ if ROB} = 0 \\ 1, \text{ if ROB} = 1 \\ 2, \text{ if ROB} = 2 \\ .3, \text{ if ROB} = 3 \end{cases}$
- Runners in Scoring Position:= $RISP_i = \begin{cases} 0, \text{ if RISP} = \text{False} \\ 1, \text{ if RISP} = \text{True} \end{cases}$
- Balls:= $B_i = \begin{cases} 0, \text{ if Balls} = 0 \\ 1, \text{ if Balls} = 1 \\ 2, \text{ if Balls} = 2 \\ 3, \text{ if Balls} = 3 \end{cases}$

- Strikes:= $S_i = \begin{cases} 0, \text{ if Strikes} = 0 \\ 1, \text{ if Strikes} = 1 \\ 2, \text{ if Strikes} = 2 \end{cases}$

Putting these definitions together we create a state vector:

$$s_i = \begin{bmatrix} I_i \\ O_i \\ ROB_i \\ RISP_i \\ B_i \\ S_i \end{bmatrix} \tag{4.1}$$

One thing to note is that we bin innings together in batches of 3. This is because we would like to distinguish between the early part of the game when both hitters and pitchers are getting settled into the game and the end of the game when there are more stressful situations on both hitters and batters. The magnitude of our possible state space $|S| = 864$ however due to the fact that not all states in our state space are reachable per the rules of baseball, the number of distinct states present in our dataset is only 540.

4.2 Calculating State Transition Matrices

Once we have calculated the game state representation for each at bat, we can use these states to learn the probabilities of transitioning from one baseball state to another. We must calculate probability distributions for both state-to-state transitions, state-event pairs, and state-pitch call pairs. The reason behind this is that often times in baseball, an event such as a single can have differing effects on the future state of the game depending on how far runners advance around the bases. This means there is not a one to one mapping between state-action pairs and the resultant state from that action. We will need to use all three in a careful manner in order to simulate

these transitions accurately and realistically.

4.2.1 State-State Transitions

In order to create this State Transition Matrix M, we can start by creating a matrix such that every row and column corresponds to one of our states s_i in our state space S giving M a shape of $|S|$ by $|S|$. Every element in M will be initialized as 0. We now loop over every pitch in every game in our dataset to create a table. Due to the fact that our dataset is incomplete and has stretches of pitches missing from some games, we take care to only look at transitions that occur between consecutive pitches. This still gives us simulation power later on as we simply treat missing pitches as data that was not observed. Because we only care about local state transitions, transitions from the current state we are looking at to the next state sequentially, it doesn't matter where these missing pitches would have been embedded in the game. After getting rid of these missing transitions, we end up with 1,865,358 total transitions. For each transition from a state s_i to s_{i+1} we augment our matrix like so:

$$M[s_i, s_{i+1}] = M[s_i, s_{i+1}] + 1 \tag{4.2}$$

By adding one to this element in M we keep track of the number of times we have seen the given transition in our dataset. After performing this calculation for all sequential pitch state transitions, we can now normalize each row to get the transition probability distribution given any starting state s_i.

$$M[s_i, x] = \frac{x}{\mathbf{sum}(M[s_i, :])} \tag{4.3}$$

This matrix M now gives the probabilities of transitioning from any state s_i to any another state s_j in our state space S at $M[s_i, s_j]$. The range of number of transitions starting from any given state s_i goes from $[13, 44532]$ with a mean of 3341. Figure 4-1

depicts the transition diagram for the state $s_i = [0\ 1\ 0\ 0\ 1\ 1]$. This translates to the game state in which we are in innings 1, 2, or 3, there is 1 out, there are no runners on base, and the count is 1-1. The transition probabilities for all possible transitions s_i to another state s_j in S sum to 1.

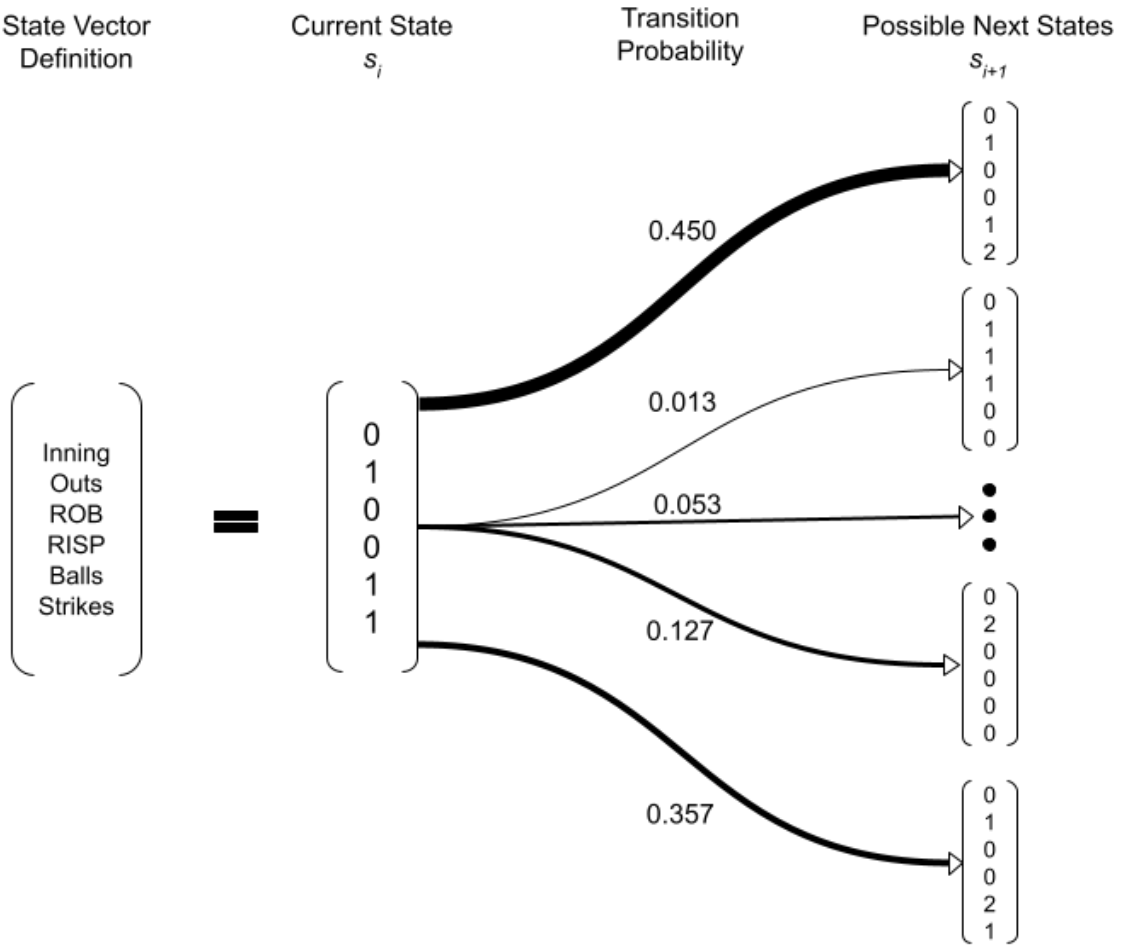

Figure 4-1: Transition probability diagram for the baseball game state [0 1 0 0 1 1] as defined in equation 4.1.

To better visualize our state transition matrix M, we will consider all states in the first three innings with 0 runners in scoring position. Figure 4-2 shows the top 20 most visited states this state subset in a chord diagram. The relative sizes of each chord corresponds to the probability of transitioning between the game state at the start of the chord to the game state and the end of the cord.

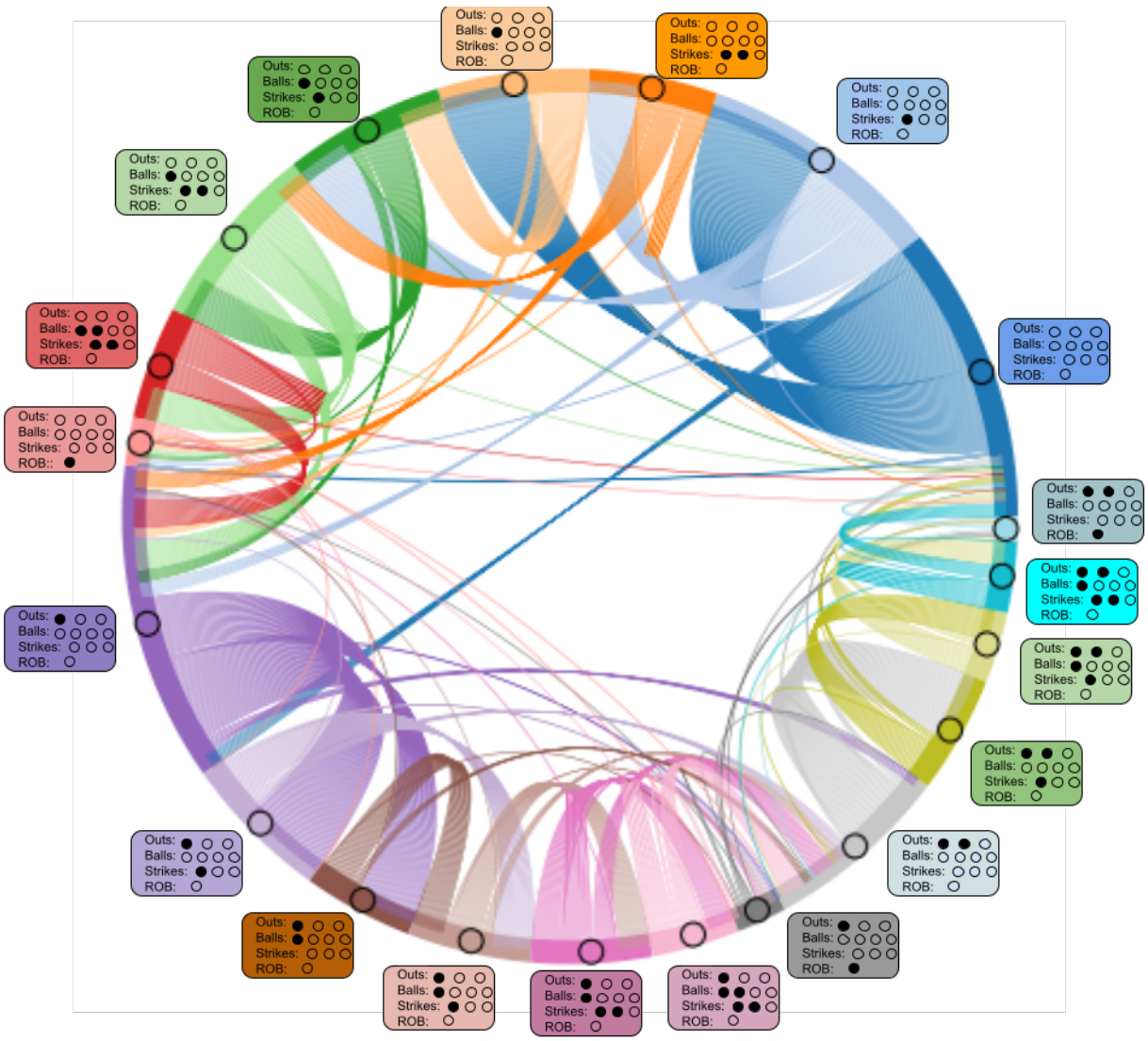

Figure 4-2: Chord diagram for the 20 most visited game states filtered by innings 1-3 and 0 runners in scoring position. Chord sizes are directly proportional to transition probabilities between states with larger chords meaning a higher probability. Each node depicts the number of outs, balls, strikes, and runners on base by filled in circles.

4.2.2 State-Event Transitions

Similarly to how we created our state transition matrix M, we can also define matrices H_f and H_t to hold probability distributions over the different hitting events that occur while in a given state and events that lead to a given state respectively. We define our event space E to be the events we will keep track of through our pipeline. The events E is comprised of describe different ways a ball can be put in play after a pitch: single, double, triple, home run, field out, and field error. The latter two respectively occur when either a hitter hits the pitch and gets out in the process of the play or the hitter hits the ball and advances successfully to a base due to a fielding mistake.

Our State-Event transition matrix H_f will have rows corresponding to all the states in our state space S, while the columns will correspond to each of the hitting events listed above in our event space E. Matrix H_t will have rows corresponding to events in our space E while the columns will correspond to states in S. These two matrices will allow us to both sample random events given a state and to sample a new state from a probability distribution given an event.

In order to account for the event that the hitter does not actually make contact with the ball, we have an **Other** event in E that we will talk about later. Much like we did in section 4.2.1, we can now loop through all pitches in our dataset, for each pitch p_i increment the counter at $H_f[s_i, e_i]$ and $H_t[e_i, s_{i+1}]$ according to the state before/after the pitch and the event that occurred during the pitch. Once we normalize, this will give us our transition matrices that will give us probability distributions for our state-event in spaces.

4.2.3 State-Pitch Transitions

Finally, we must account for the situations in which a pitch is thrown and the batter does not hit the ball into fair play. For these situations, we create a transition matrix T like we do in section 4.2.2 except the columns are now pitch call events in the space P. These sorts of events include: called strikes, swinging strikes, balls, foul balls, and hit by pitch.

Figures 4-3, 4-4, 4-5, and 4-6 allow us to visualize these matrix definitions. Once we have created our transition matrices M, H_f, H_t, and T we can now use them to define a method for formally simulating the states and events that occur in a baseball game.

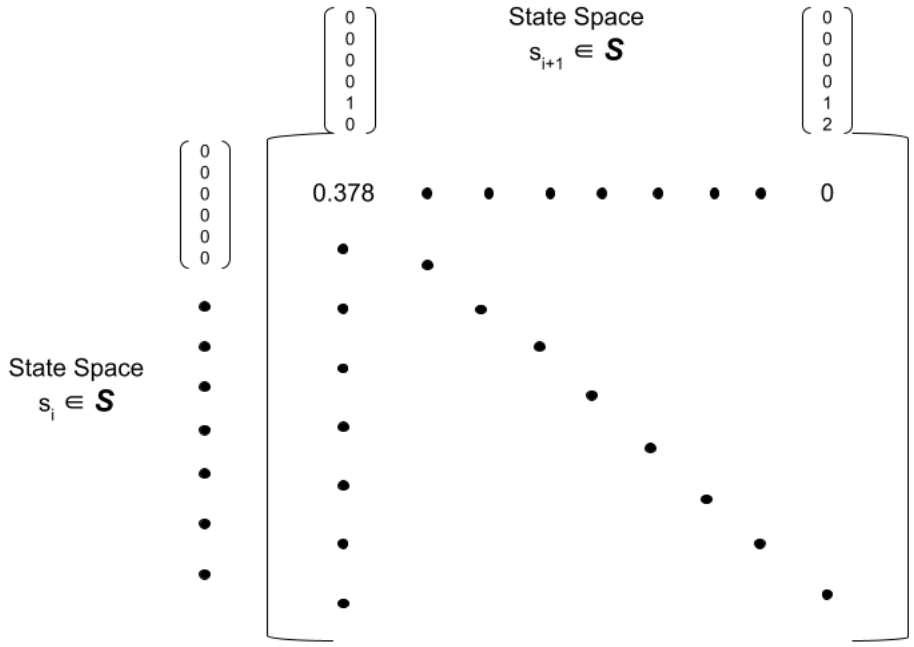

Figure 4-3: Definition of Matrix M

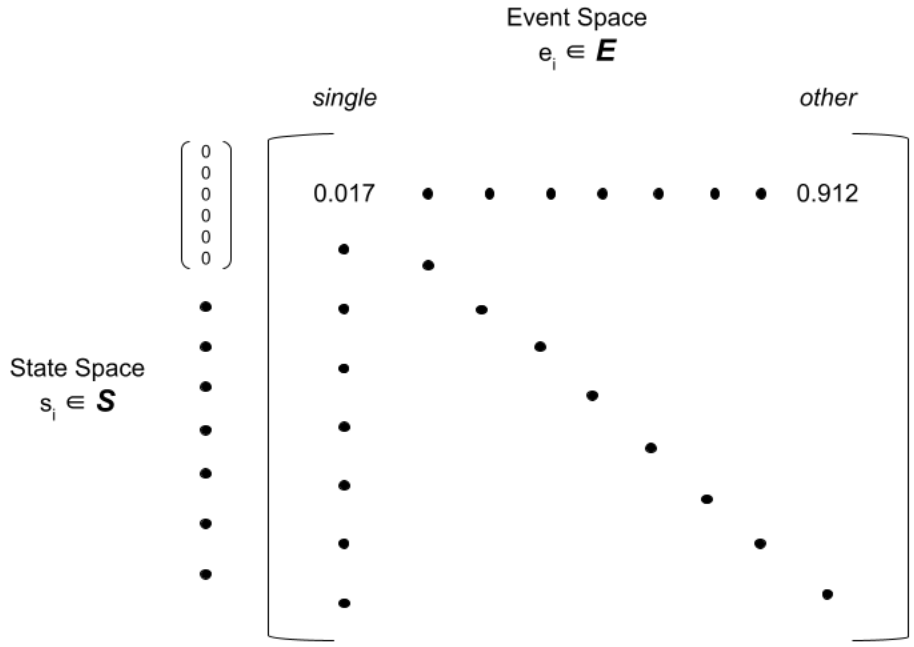

Figure 4-4: Definition of Matrix H_f

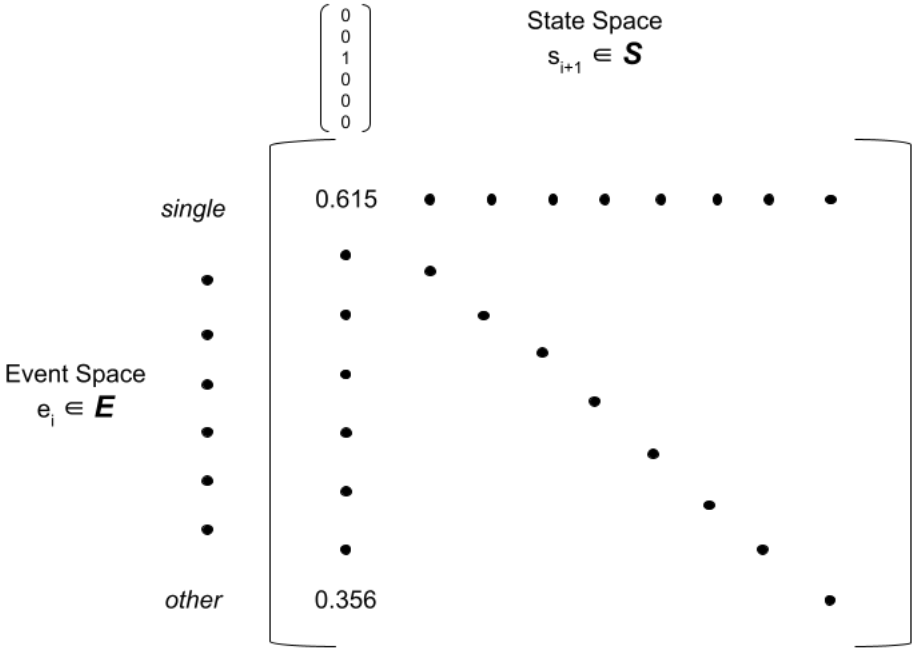

Figure 4-5: Definition of Matrix H_t

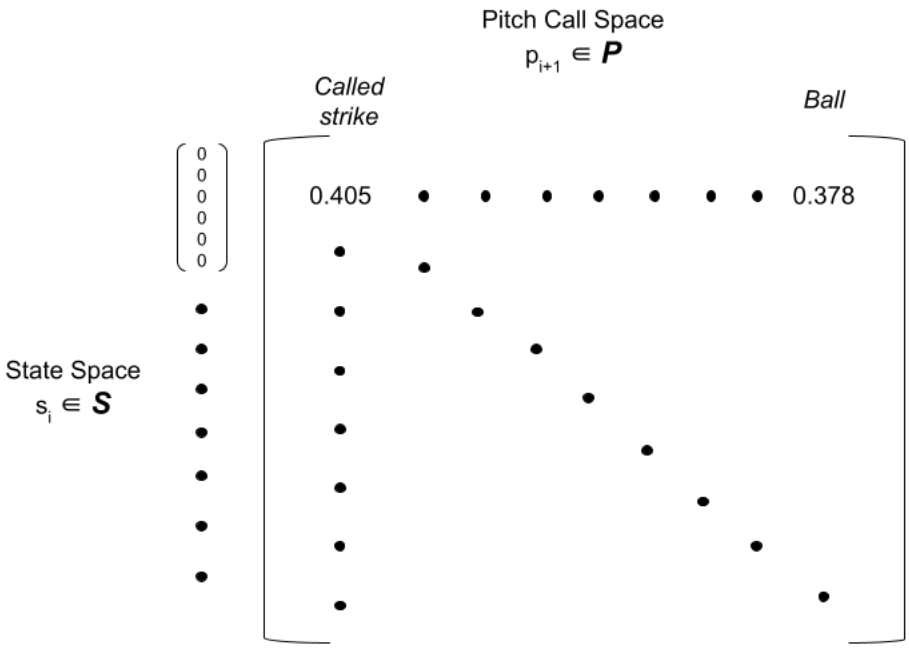

Figure 4-6: Definition of Matrix T

Chapter 5

Monte Carlo Counterfactual Simulations

Ultimately, the goal of this pipeline is to be able to quantify how the key statistics in a baseball game (number of pitches, balls in play, hits, and home runs) change if the MLB were to switch to an automated umping system. In order to do this we will first look at the game state we are in when the mistake is made, correct the mistake by switching to a new game state, and then extrapolate out the rest of the inning by simulating the future game transitions multiple times and averaging.

5.1 Monte Carlo and Confidence Intervals

Monte Carlo methods allow us to model statistical results through the use of randomness. Specifically in the case of a baseball game, we are dealing with a class of Monte Carlo methods known as a Markov Chain Monte Carlo (MCMC) problem. For these chains, the posterior distribution for transitioning to a state s_i depends only on s_{i-1} [17]. This fits exactly with the transition matrices we defined section 4.2 as the probability distributions in each row depends solely on the previous state the game is in.

A key parameter to tune in Monte Carlo simulations is the number of iterations τ. For our methods, τ will correspond to the number of times we simulate a correct half

inning. Depending on the value chosen for τ, one must be concerned about whether or not the Monte Carlo simulation will converge to represent the actual sample mean and variance of the random variable we are trying to estimate (baseball statistics in our case). For this work, we will focus on generating results that fall within a 95% confidence interval, meaning the true parameter value we are estimating (expected hits, home runs, or pitches) will fall within this confidence interval range 95% of the time.

Define our confidence intervals for each of our Monte Carlo simulations with a sample size of τ as follows. Because we do not know the mean μ or variance σ^2 of our Monte Carlo variables and each sample is sampled independently, we can define:

$$\bar{X} = (X_1 + \cdots + X_\tau)/\tau \tag{5.1}$$

$$V^2 = \frac{1}{\tau - 1} \sum_{i=1}^{\tau} (X_i - \bar{X})^2 \tag{5.2}$$

Where X is our sample mean and V^2 is the sample variance. We can then define

$$T = \frac{\bar{X} - \mu}{V/\sqrt{\tau}} \tag{5.3}$$

where T has what is called a Student's t distribution which is a continuous probability distribution that arises when estimating the mean of a normally distributed population where the sample size is small and the standard deviation is unknown [14]. Letting c be equal to our 97.5th percentile distribution (a z-score of 1.96), we get:

$$Pr\big(-c \leq T \leq c\big) = 0.95 \tag{5.4}$$

We can then use this and equation 5.3 to generate a 95% confidence interval for our population mean μ as follows:

$$Pr\big(\bar{X} - \frac{cV}{\sqrt{\tau}} \leq \mu \leq \bar{X} + \frac{cV}{\sqrt{\tau}}\big) = 0.95 \tag{5.5}$$

Finally, once we run a single Monte Carlo simulation and observe our sample, we can substitute in the respective values to obtain the 95% confidence interval range:

$$\left[\bar{X} - \frac{cV}{\sqrt{\tau}}, \bar{X} + \frac{cV}{\sqrt{\tau}}\right] \tag{5.6}$$

This definition will help us to define and evaluate our final Monte Carlo method results as we look to compare them to the actual aggregated MLB results.

5.2 Performing Monte Carlo Simulations

Now that we have laid the groundwork for the feasibility of using Markov Chain Monte Carlo Methods with reference to our problem formulation, we can look at the specific algorithm used to effectively correct an umpire's mistake and simulate the results following the change.

The first assumption we make is that individual half innings are independent of each other. Both hitters and batters are going to try and perform to the best of their abilities and because the only result from one half inning that carries over into another is the score, we can assume our game state resets when a new half inning starts. This means that statistical results in innings without mistakes can be used as if it was played under our automated umpiring model. In order then to look at how games change under this model, we only need to simulate half innings that have a mistake in them.

In any half inning, the game state for any pitch p_i is dependent on the sequence of pitches $p_0, p_1, ..., p_{i-1}$ leading up to it. This means that for a sequence of pitches defining a half inning, wherever there is a mistake, all proceeding pitches are possibly affected and therefore must be simulated. Moreover we will define a sequence of

consecutive pitches seq where each pitch p_i is defined by `(PitchNumber, isMistake)` where `PitchNumber` is the sequential number of the pitch in the inning and `isMistake` is a classification of the pitch as correctly called given our umpiring model as defined in 3.1. A Monte Carlo simulation's starting pitch i_{start} must be the very first mistake in the half inning, namely:

$$i_{start} = argmin(p_i.\texttt{PitchNumber} \text{ if } p_i.\texttt{isMistake} == True) \tag{5.7}$$

Once we have found the first pitch with a mistake, in order to estimate the statistical results we care about we can sum the results from pitches p_0 to $p_{i_{start}-1} = h_i$, correct the pitch call at $p_{i_{start}}$ to $p_{i'}$ and then use our Monte Carlo method to find the estimated results following the correction of the mistake until the end of the half inning $m_i = Monte(p_{i'})$. This will give us our estimated results $h_i + m_i$.

Next, we define how we use our transition matrices M, H, and T that we defined in section 4.2 to move from state to state during simulation. Algorithm 1 lays out this overall method before we dive into the Monte Carlo Simulation aspect:

To simulate through a half inning as shown in helper function $monteCarloSim()$ we follow these general steps:

1. Generate a new state s_{new} from our input game state s_i depending on whether the pitch call was mistakenly called a ball or strike

2. Sample a random event e_i weighted according to the posterior distribution $H[s_i]$.

   (a) If $e_i =$ `Other` as defined in 4.2.2, we have randomly sampled from space P. Randomly sample a new random pitch call from the posterior distribution $T[s_i]$ and define it as e_i.

   (b) If $e_i \neq$ `Other` then we have chosen a hitting event from our event space E and can use it.

```
Data: MLB Pitches dataset with ball tracking and game metadata
      information
Result: Estimated game statistical results after correcting umpire mistakes
for game in games do
    for half_inning in game do
        mistakes = calculateMistakes(half_inning);
        first_mistake = argmin(mistakes);
        for sorted(pitch) in half_inning do
            if pitch.pitchNumber != first_mistake then
                stats += accumulateStats(game, pitch)
            end
            else
                stats += monteCarloSim(game, pitch) break;
            end
        end
    end
    saveResults(game, stats)
end
```

Algorithm 1: Overall Simulation Algorithm: finds half innings with mistakes and simulates following the first mistake in the inning

(c) Accumulate a counter for the sampled event/pitch call to keep track of the number of occurrences in this iteration

3. Extract our State-State Transition distribution $M[s_i]$

4. Eliminate all states s_{i+1} from $M[s_i]$ s.t. $s_{i+1} \notin \{H_t[e_i]\}$

5. Renormalize and randomly sample a new state s_{i+1} from the probability distribution $M[s_i]$.

6. Set our current state $s_i = s_{i+1}$.

7. Repeats steps 2-6 until the number of outs in our game state s_i is equal to 3 (end of the half inning).

8. Log the statistical results for the given monte carlo iteration.

9. Repeat steps 2-8 for τ iterations.

10. Average the final results to get the expected statistics for the simulated half inning after correcting the mistake at pitch p_i.

One of the key steps in this simulation approach is step 4. During this step, we use our posterior distributions we computed for which events lead to which states in order to generate valid state-state transitions. After sampling our event, we find all the states that are never transitioned to following the sampled event and get rid of them as possible next states. This assumption relies on the fact that we have a large enough dataset that fully represents all possible event transitions. However, in order to ensure we were only performing valid state transitions, this rejection sampling strategy was the best way to leverage the existing historical data without guessing what the state could look like following an event.

In order to then extrapolate this Monte Carlo strategy to a game we first iterate through a game and log the actual results. Then, we re-loop through all 18 half-innings in the game, and use algorithm 1 to simulate the number of hits we would expect to see given an automated umpiring system.

5.3 Average MLB Game

One of the most interesting questions we can answer using this Monte Carlo method is "what does the average MLB game look like?" This is equivalent to running our Monte Carlo method with a starting state of all 0's except for the inning state which would correspond to whichever inning we were simulating and summing up the 18 results. Figure 5-1 shows different statistical comparisons between the average MLB game in the whole dataset and a completely simulated game using our Monte Carlo method with $\tau = 2000$. For all three categories, the MLB Dataset has a much higher standard deviation, owing to the fact that because we are sampling from aggregated posterior distributions of state and event transitions, we see much fewer outliers in our Monte Carlo simulations because we use a large τ value and average our results over all iterations.

Figure 5-2 lets us analyze the effect that τ has on the results we obtain while simulating the average MLB game. All three statistical plots jump around initially before leveling off and rising to their respective stable values. We chose to use $\tau =$

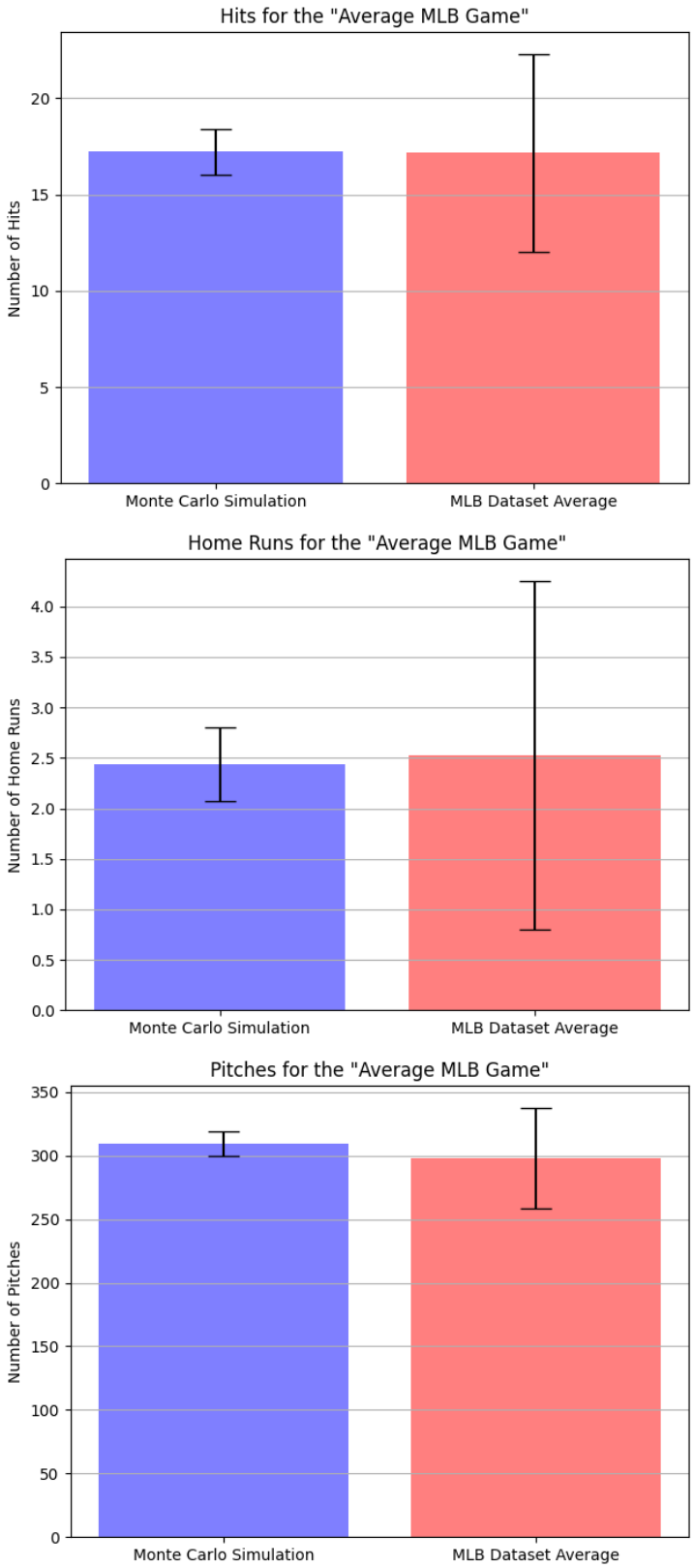

Figure 5-1: Comparison of number statistical results in a Monte Carlo simulated game compared to the average MLB game. The black error bars depict standard deviations.

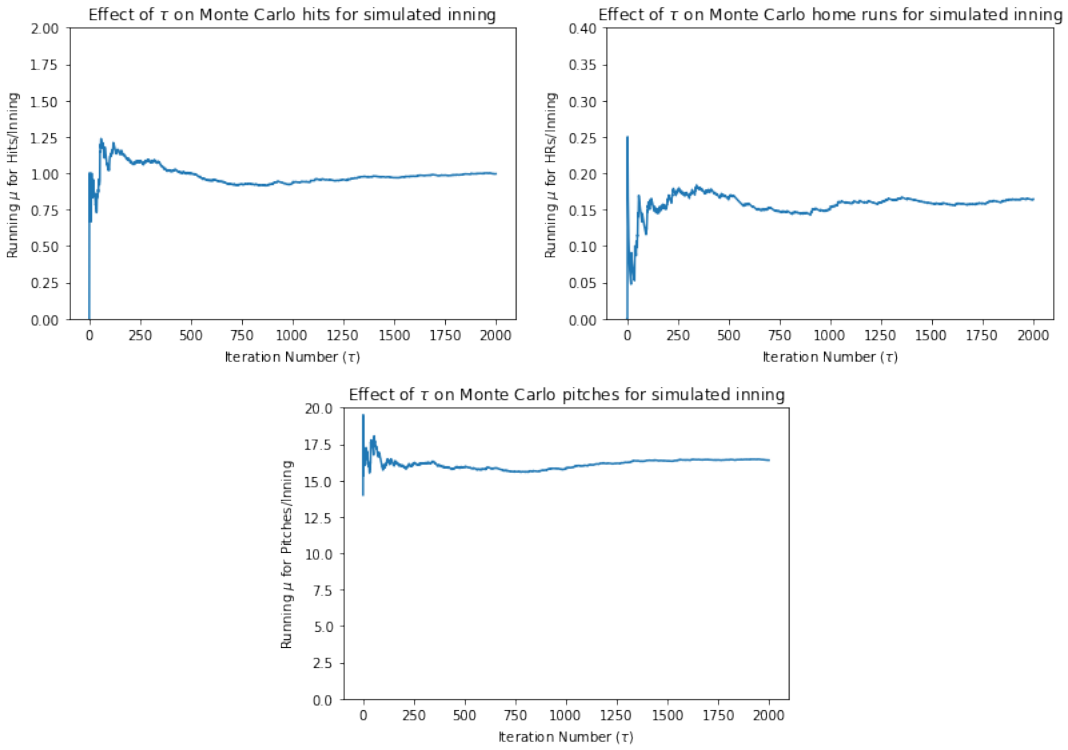

Figure 5-2: Visualizing effects of different iteration numbers τ when simulating a Monte Carlo inning.

2000 when running our model on the MLB dataset as we believed this number of iterations allowed convergence of values while also being feasibly possible to run. Given we have 7276 games in our dataset, we will simulate on the order of 18 half innings per game (possibly more or less depending on the absence of mistakes or surplus of innings in a game that lasts longer than 9 innings. This is $\approx$ 131000 simulations. For the scope of this thesis, we needed to be able to run all of these simulations in a timely manner in order to analyze the results and thus 2000 was an upper bound on the number of iterations we could run while still maintaining feasibility.

Chapter 6

Evaluation

After applying our mistake classification method, creating our game state representation transition matrices, and defining a robust model for using both in tangent to fix classified mistakes using a Monte Carlo simulation, we can now look at evaluating our results to draw conclusions about this problem space. We will use multiple figure types in order to visualize different patterns between the resulting statistics. We will compare results both on a dataset-wide level as well as look at trends between different innings and home-away status of the teams.

6.1 MLB Dataset Comparison

To begin, we can first look holistically at how our Monte Carlo simulated game results compare to those of the MLB dataset aggregations on a game by game basis. These results will be able to show us how the game stats we are focused on change as a result of switching our umpiring system. These differences can help us determine whether MLB-driven efforts to enhance the in person viewing experience would have the desired or possibly undesired effect. Figures 6-1, 6-2, and 6-3 show histograms of the relevant results for all $7,726$ games in our MLB Dataset. The green bars are plots of the results from each game after fixing umpire mistakes and simulating through the rest of innings with mistakes. We also visualize the means as dashed lines for both groups of statistics. For all three major categories, our Monte Carlo results

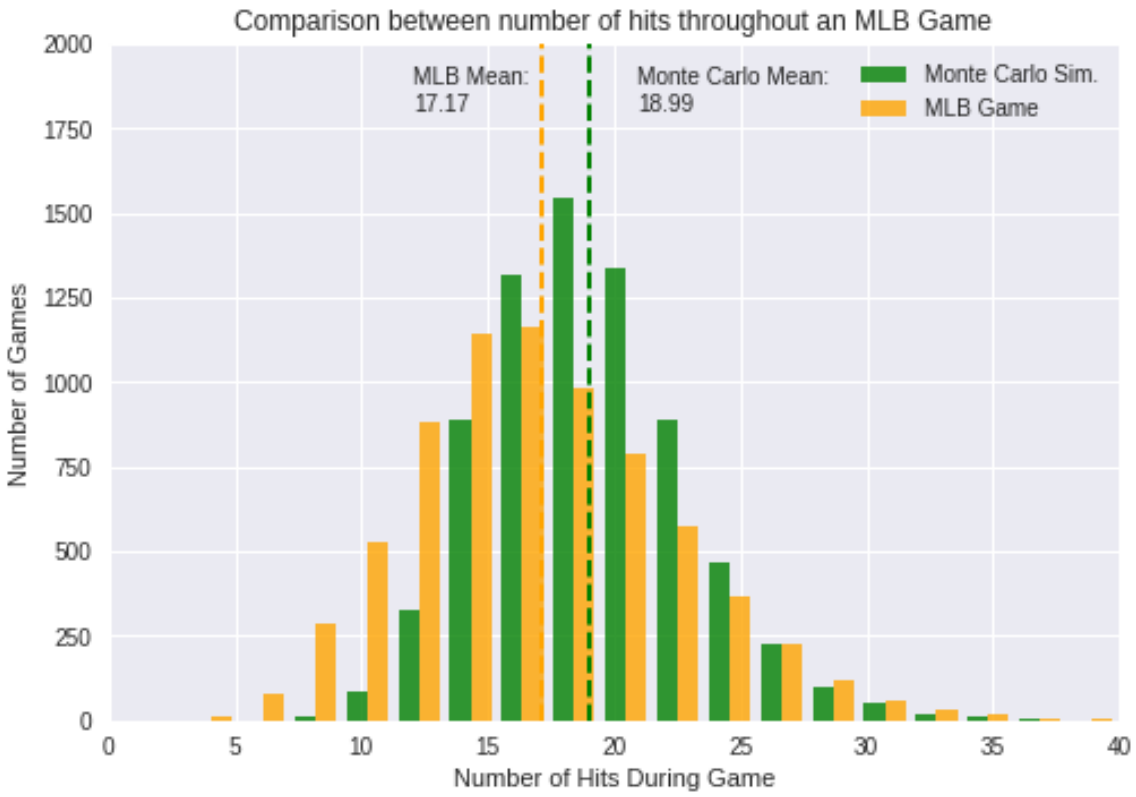

Figure 6-1: Histogram charting number of **hits** in Monte Carlo fixed games vs. MLB games.

seem to have higher averages. This makes intuitive baseball sense, as when referring back to Table 3.3, we can see the majority of calls that are mistakes are "Called Strikes". This means when we correct the game state by taking away a strike and adding a ball, the batter is now at a more favorable count. In the long run, we would expect more hits/pitches/home runs to stem from pitch counts that are better for the batters. We also can see that on the contrary to the hits and pitches plots, in Figure 6-2 the Monte Carlo distribution has a slightly right skewed distribution while the MLB Game distribution is more uniform. This could be one possible cause of the differences we see in means as there are ≈ 600 more games in the lowest bin for MLB Games, bringing down the mean. This could be because MLB games can only have an integer number of home runs.

To further dive into how the games changed due to automated umpiring, we can also turn to standard deviations and our confidence intervals as defined in Section 5.1.

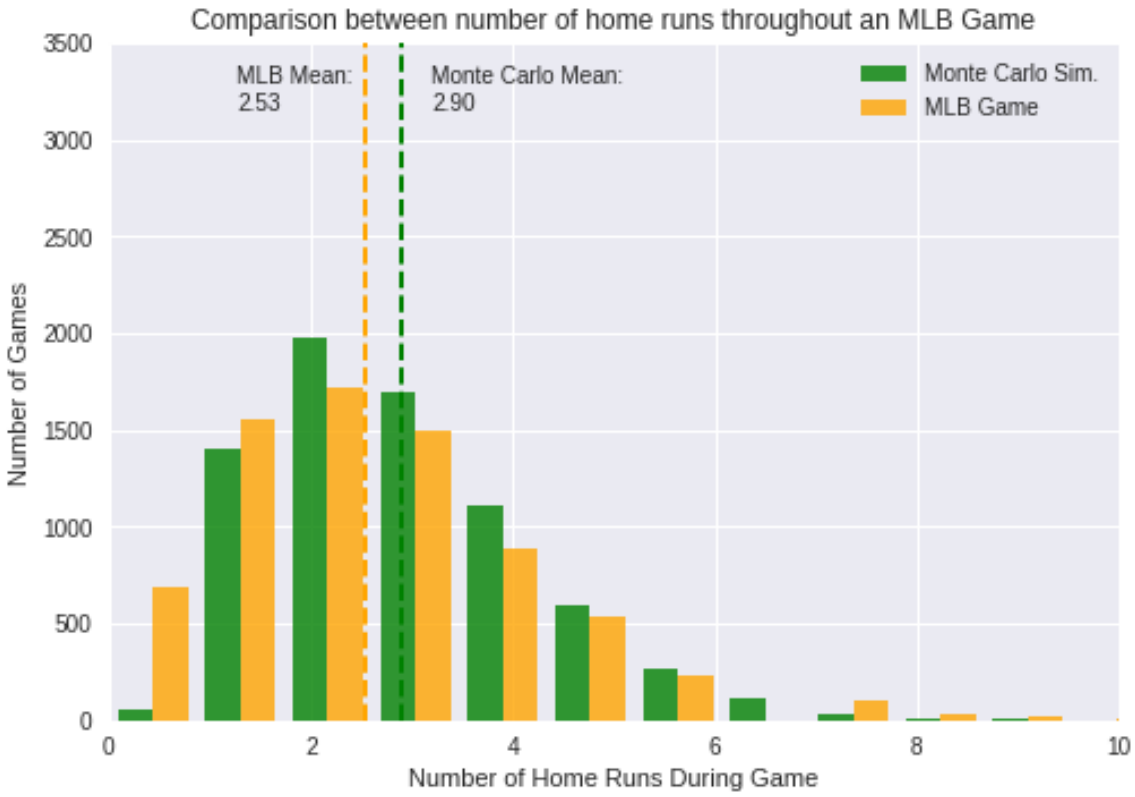

Figure 6-2: Histogram charting number of **home runs** in Monte Carlo fixed games vs. MLB games.

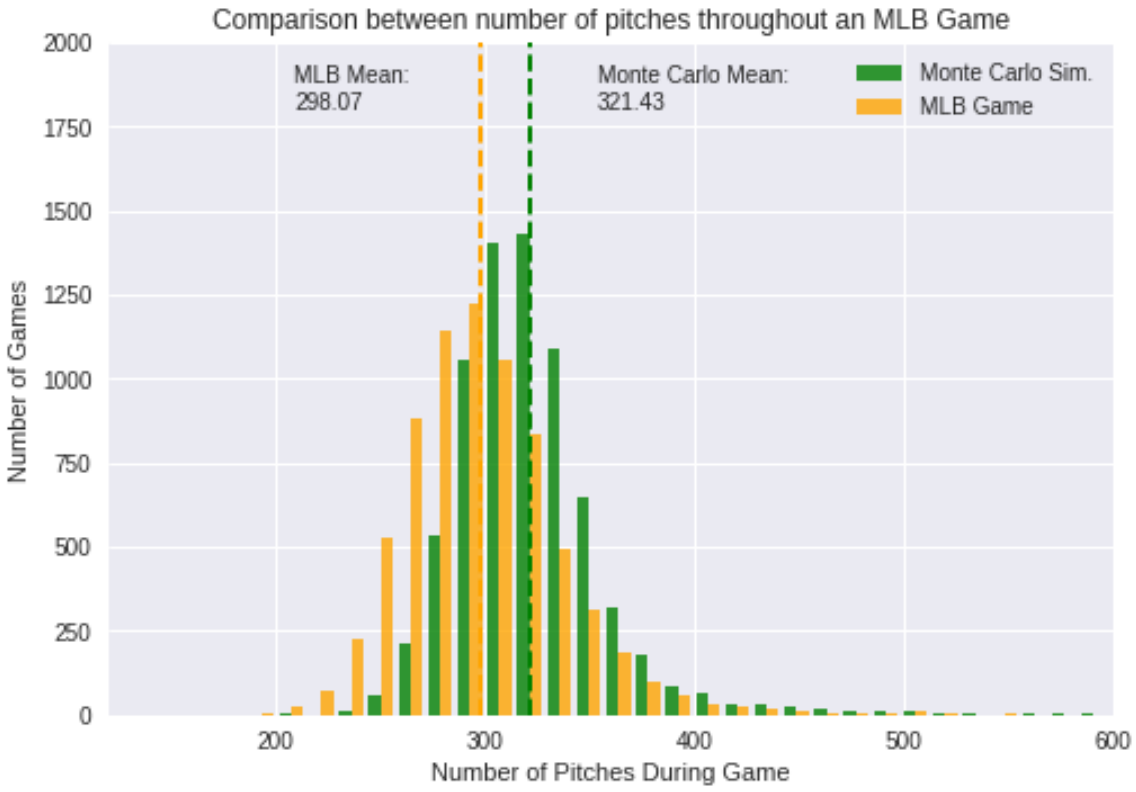

Figure 6-3: Histogram charting number of **pitches** in Monte Carlo fixed games vs. MLB games.

More specifically, for each statistical category, we compared the means μ for both our Monte Carlo simulated games and the actual MLB games using the following formula:

$$\% \text{ Change} = \frac{\text{Monte } \mu - \text{Actual } \mu}{\text{Actual } \mu} \tag{6.1}$$

This allowed us to compare the percentage change in all categories with respect to our baseline results for all MLB games. Figure 6-4 lets us visualize our findings. We plot a dashed line at 0 to show the baseline for zero change between both game models. Each statistical point also shows upper and lower 95% confidence intervals over the true mean of our Monte Carlo results. For both hits and home runs we can see that there was around a $10 - 15\%$ increase in both stats as a result of an automated umpiring system. Pitches also showed an increase with a 7.8% increase as well, although we can see the error bar does drop below 0. This shows that in the process of correcting mistakes, although we more often than not create more favorable hitting counts for batters, our model also can have the opposite effect of reducing the number of pitches in the game.

6.2 Home vs. Away

In baseball, there is a saying commonly known as "home-field advantage". This refers to the often imagined but sometimes statistically proven idea that the Home team (team hosting the game in their stadium) has a competitive advantage due to a multitude of factors. These factors often include fan attendance and crowd noise, batting in the bottom of the inning, or simply the comfort of playing in a stadium you are more familiar with. People also speculate about whether umpires make decisions in favor of the home team in order to avoid backlash from heckling fan bases. This is not supposed to be the case as umpires are supposed to call a baseball game fairly for both teams. Using the model we have defined, we can look at how our statistical results change with regards to the status of the team the umpire is making the calls

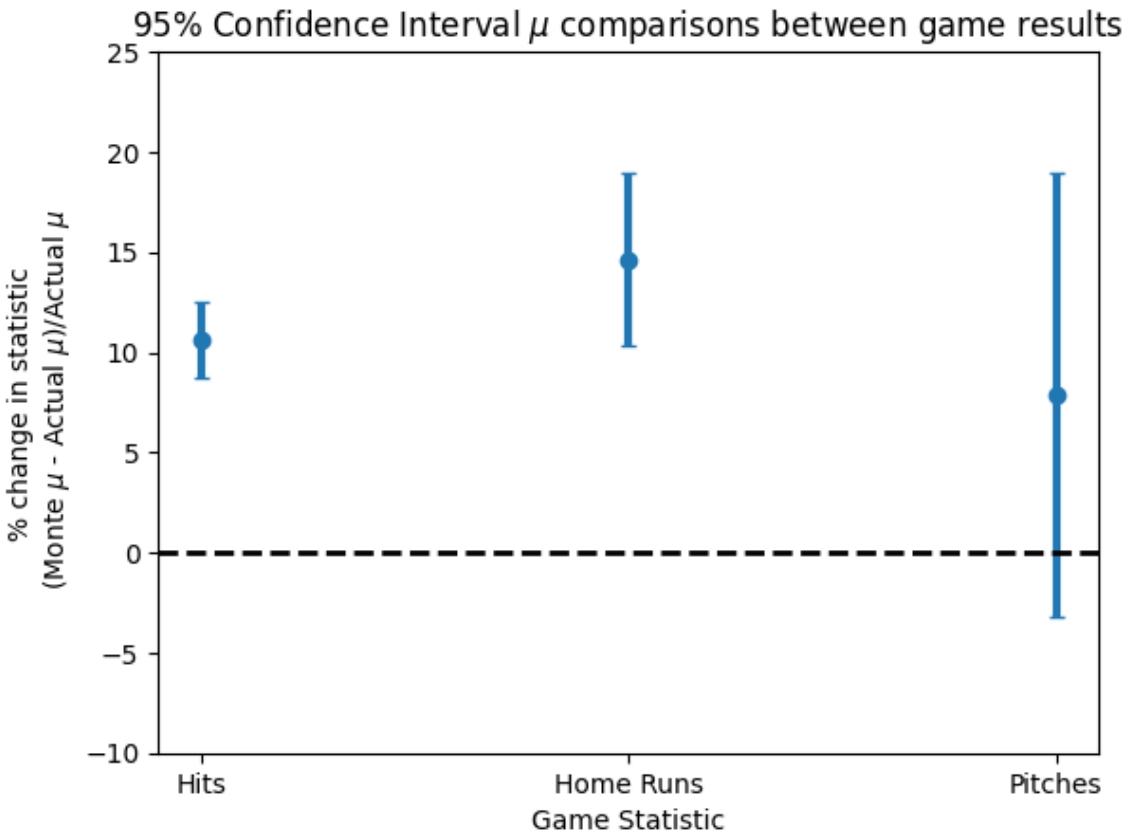

Figure 6-4: Analysis of the result mean differences for each statistical category.

for (Home vs. Away). Figures 6-5, 6-6, and 6-7 allow us to break down our results from earlier into these two categories.

From both our **Hits** and **Home Runs** distributions, we can see there is little to no difference between Home and Away teams. There is a slight difference between Away and Home teams when looking at the **Pitches** distribution with Home teams pitching slightly more pitches than away teams for both actual games and our Monte Carlo simulations. This could be due to the fact that if a Home team enters the bottom of the 9th inning winning already, the game is immediately over and the Away team does not have to pitch that half inning, leading to slightly fewer pitches over all games.

We can also visualize these same results by looking at the differences between our Monte Carlo simulations and the actual games on a game by game basis (Figures 6-8, 6-9, and 6-10). We can see that although there is skewing of the distributions

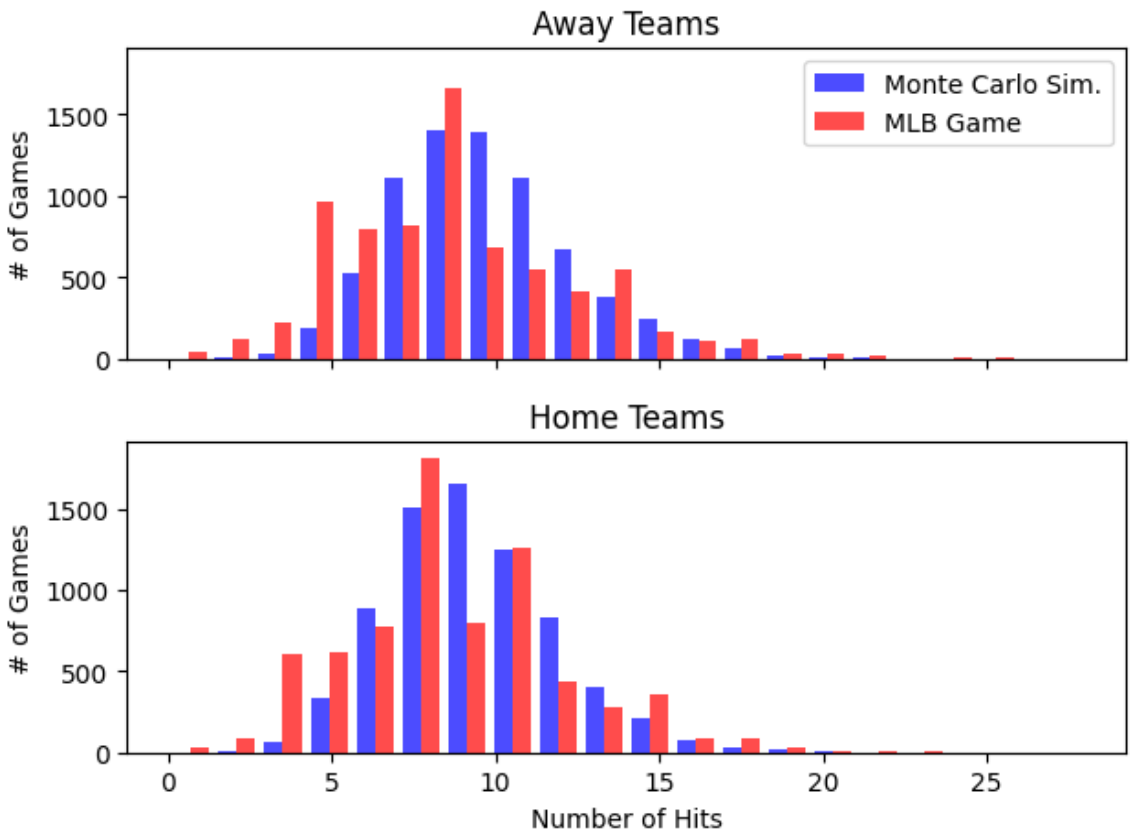

Figure 6-5: Histogram charting number of **Hits** in Monte Carlo fixed games vs. MLB games for **Home vs. Away** teams.

horizontally, this is indicative of differences between our model's results and the game results. Since both Home and Away distributions skew in the same way, we don't have any reason to suspect that umpires are making decisions differently depending on the Home-Away status of a player. This is both good for the game of baseball to know as well as good for an automated umpiring system in the future as using this model would not have any unintended effects on MLB games.

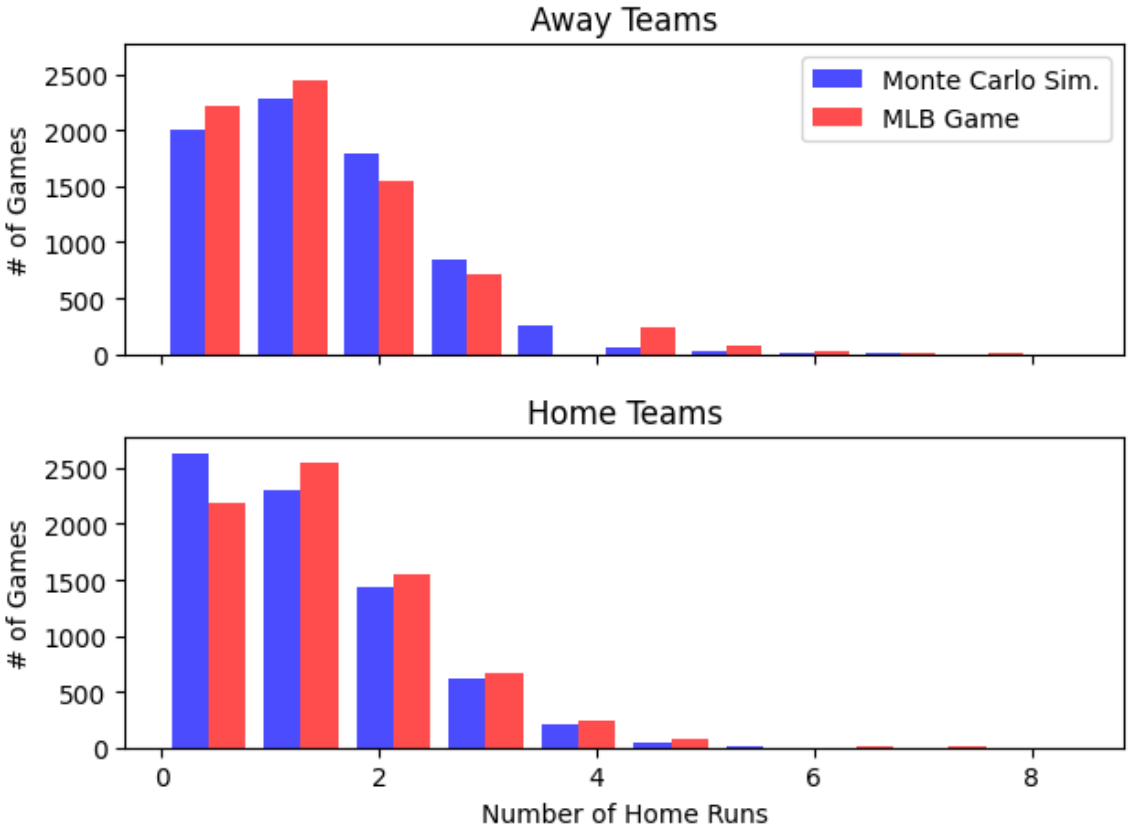

Figure 6-6: Histogram charting number of **Home Runs** in Monte Carlo fixed games vs. MLB games for **Home vs. Away** teams.

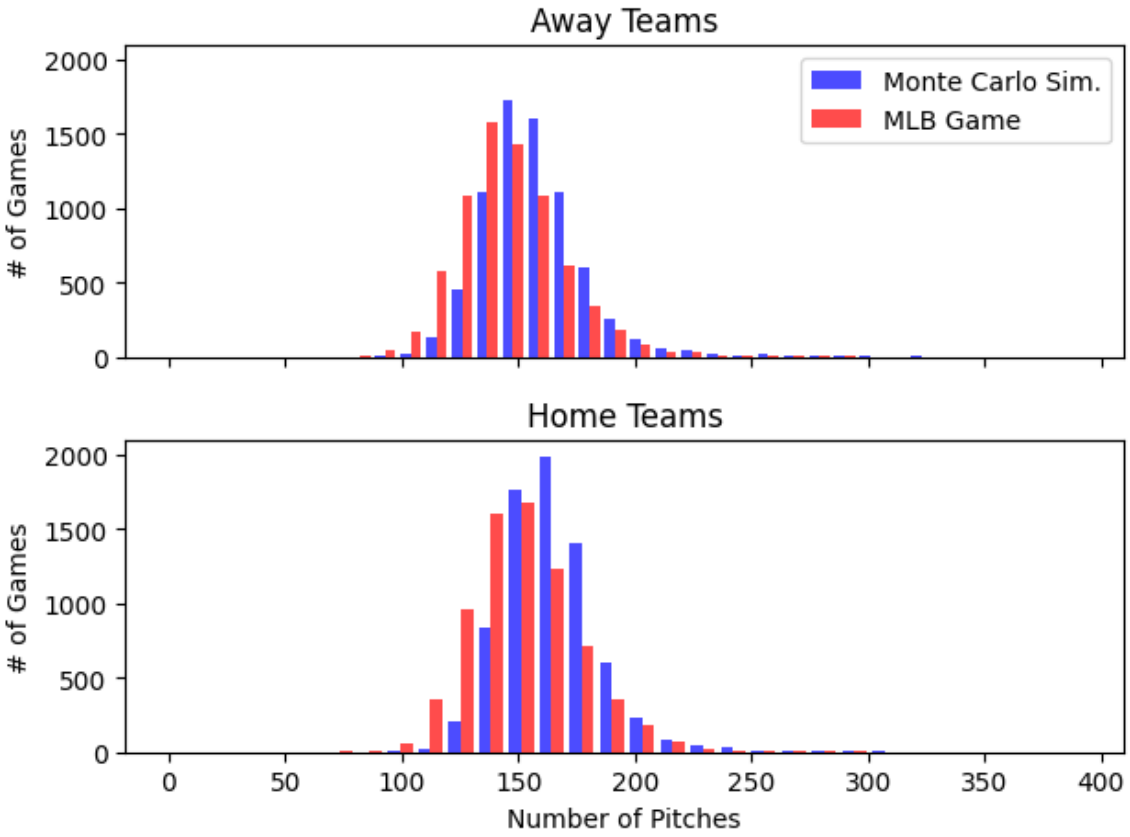

Figure 6-7: Histogram charting number of **Pitches** in Monte Carlo fixed games vs. MLB games for **Home vs. Away** teams.

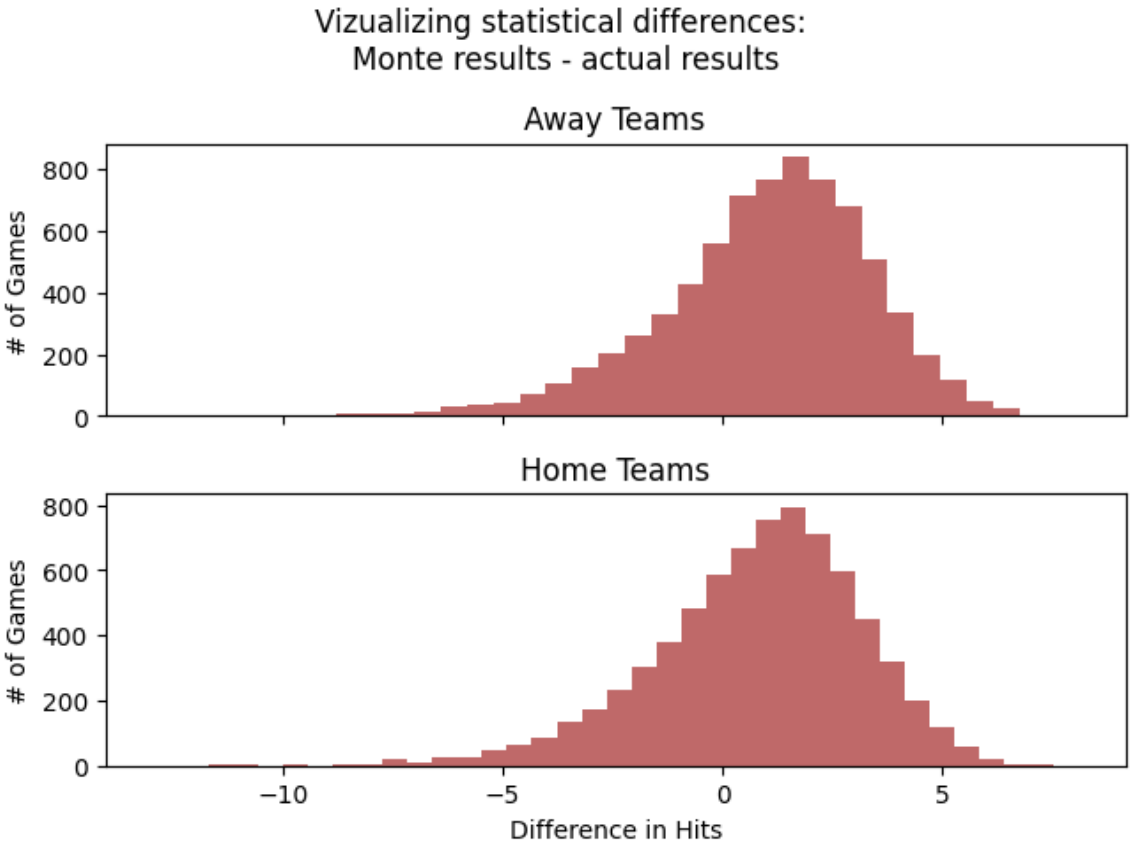

Figure 6-8: Histogram of differences in **Hits** between Monte results and actual results for **Home vs. Away** teams.

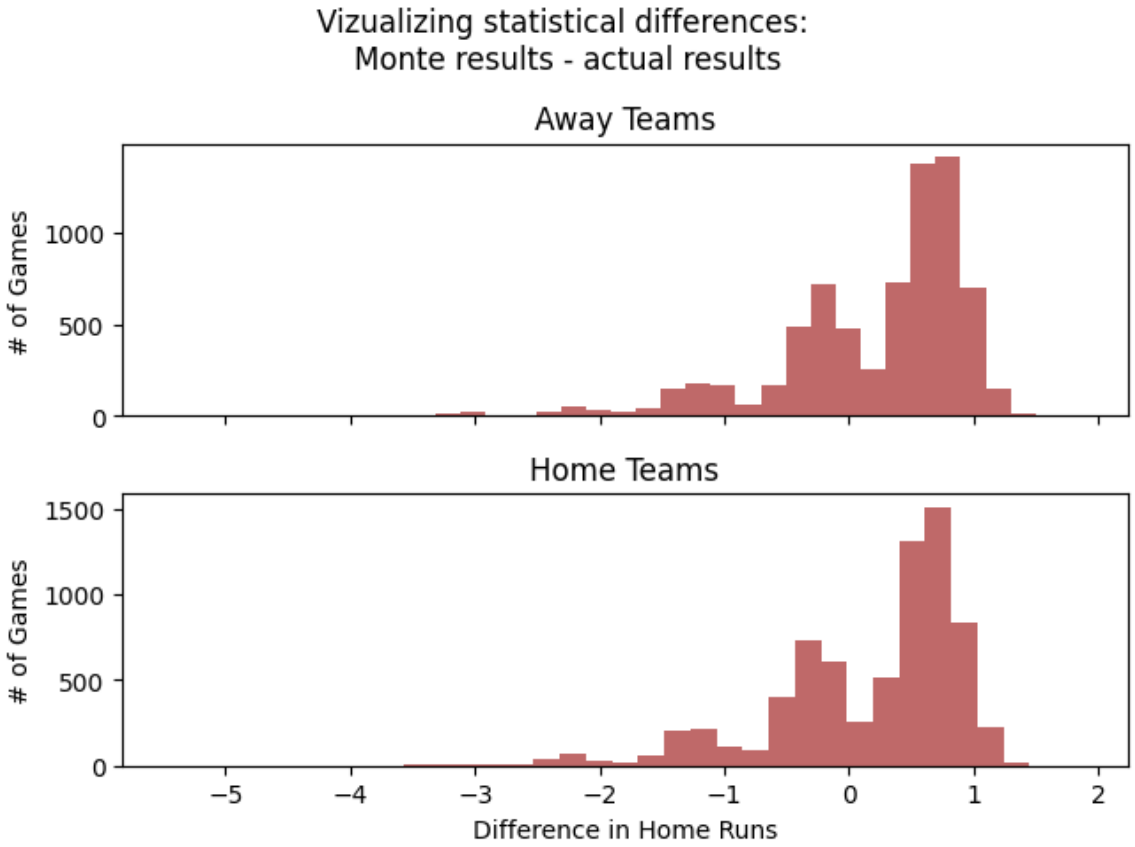

Figure 6-9: Histogram of differences in **Home Runs** between Monte results and actual results for **Home vs. Away** teams.

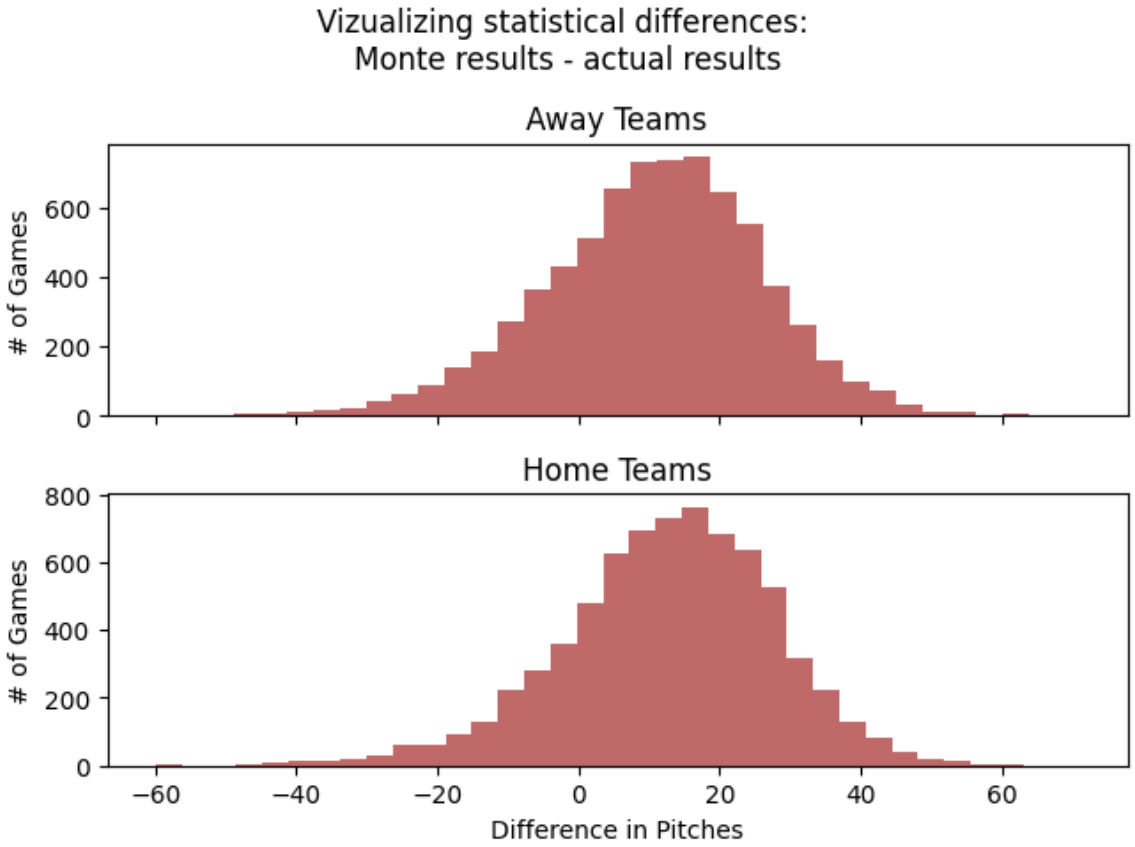

Figure 6-10: Histogram of differences in **Pitches** between Monte results and actual results for **Home vs. Away** teams.

Chapter 7

Conclusion

In this chapter we will present some possible areas to expand on the techniques presented in this thesis. We also look at the key insights to take away from the results of our work.

7.1 Future Work

There are multiple areas for possible improvement of our game simulation method we have defined in this work.

The first area would be further expansion of our game state representation to fully encapsulate all aspects of a baseball game. This could include adding features such as scores for each team, whether we are in the top or bottom of an inning, or specifics about which runners are on which bases. This would provide a more comprehensive game state and allow for further specificity in game state transition matrices. On the flip side, this would also greatly increase the state space size and in turn more data would be needed in order to have a non-trivial amount of observations for each game state. In order to increase the dataset size, one could look into using baseball data that pre-dates 2017.

Next, given an expansion of the game state representation to include runs for each team, including the analysis of runs scored as a statistical category could be a further improvement on detailing the effects this model would have to the game. By

analyzing runs scored, one would be able to look at both changes in runs as well as changes in possible win-loss records for teams due to those run changes. This work details how in game events could change as a result of an automated umping model, however an analysis that identified both how in game events and in turn game results changed could influence whether or not the MLB adopts this methodology.

One oversimplification that this model makes when defining the game events in our event/pitch spaces E and P is the types of balls in play/pitch events that are possible. Events such as double plays, pass balls, and runners stealing bases are all things that happen over the course of a baseball game that change the game state. For the sake of this work, we simplified our model not include these edge case events. An expansion of the Monte Carlo simulations to include these events would give more predictive power to the true results of baseball games as these events change game results in ways that this model is unable to quantify in its current form.

In all types of sports, the one factor that drives success is player performance. Accurately predicting player performance, either for the sake of adding new players through a draft or trades or identifying players to get rid of, is a challenge faced by all teams in the MLB. Although the findings in [15] are focused on predicting performance for players transitioning from one professional league in Japan to the American MLB, the methodologies still apply generally to predicting a baseball player's statistical performance. One way of doing this is through clustering methods. Given information about similar hitters in the MLB and how they performed throughout the year, one can make predictions about how that hitter will do in an upcoming year. This work treats all hitters and pitchers as equivalent when generating our game state transition probabilities. Those involved in baseball know however that not all hitters and batters are the same. Some pitchers are great when they're ahead in the count while others are just not as good pitchers for whatever reason. Some batters hit for lost of home runs but strike out often while others consistently get on base due to hits or walks but don't hit as many home runs. Transitions between game states would change depending on what hitters or pitchers are currently at the plate/mound. For this thesis, we started doing some statistical based k-means clustering on player performance in

order to more robustly aggregate game state transitions based on equivalent pitcher-batter match ups. Due to the limitation in our dataset size, we were unable to draw reasonable conclusions from the game state transition matrices due to the fact that we did not observe enough hitter-pitcher match-ups between across all clusters to create accurate probability distributions. With the addition of more data, implementing a clustering method for similarly skilled players could result in game state transitions and simulation results that more accurately depict an MLB game.

7.2 Key Insights

Our Monte Carlo based simulation method was able to produce realistic results that we can draw conclusions from to answer the underlying question proposed in this work: how would MLB baseball games change with the implementation of an automated umping model. We were able to demonstrate how a possible umping model would operate with respect to the definition of the strike zone in order to classify umpire decisions as correct/incorrect. We also defined a representation for the state of the game that allowed us to generate transition matrices in order to simulate a natural and viable game progression. Finally, we were able to use the prior two contributions together to define a Monte Carlo Method for approximating the statistical results of a game given the correction of umpire mistakes throughout. In the process of developing this pipeline, we were able to draw several important insights about the typical MLB game that deserve reflection.

First, through our definition of an automated umpire pitch classification model, we were able to identify a trend in the types of mistakes that umpires make. Although most mistakes tend to be on pitches that lie around the border of the zone, most of the calls that were classified as mistakes were called strikes by the umpire. Taken in the context of a game, this is very important as it means when umpires make mistakes $\approx 89\%$ of the time it is in favor of pitcher as it leads to another strike in the count or even a possible strikeout.

We also were able to define a contextually accurate game state transition model

that relied on real-game transitions. Rather than trying to enumerate possible movements from one state to another and manually removing those that were impossible, we relied on our dataset to do this for us. This gave us realistic transition probability distributions based on historical game data. One could extrapolate this methodology for any sport in which you can reasonably represent the game state and for which you have a dataset large enough to generate transition probabilities.

Through the use of our Monte Carlo simulations, we identified small yet non-trivial changes in the statistical results of baseball games. Our automated umpiring model suggests automated umping may lead to increasing values in all three major categories we looked at (hits, home runs, and pitches). Taking on the lens of a casual baseball fan, these findings are both good and bad. An increase in hits and home runs means more runs and balls in play which would increase the excitement for most. On the other hand, increasing the number of pitches would lead to an increase in duration of the game with all else being equal.

These statistical findings are meant to be an unbiased view into the effects that an automated umping system could have on the game of baseball in the MLB. Our methods are by no means comprehensive of the intricacies of the game, however the results obtained were reached through the analysis of **historical** data and can serve as an **accurate** first step towards deciding whether this sort of technology should be a part of the future of baseball.

Appendix A

Software Packages

All of the code used to perform the analyses throughout this work was written in Python. This section highlights some of the major packages used to simulate and visualize the results of using an automated umpiring model during MLB games. This list is not extensive and only serves to illustrate the main high level tools that were utilized.

1. **matplotlib**: data analysis visualization library [3]
2. **numpy**: efficient mathematical computational library, also used as random number generator [6]
3. **pandas**: fast and efficient data analysis/manipulation tool [7]
4. **scikit-learn**: used for k-means clustering of hitters, predictive data analysis tool [8]
5. **seaborn**: Package for enhanced visualization and formatting, built on top of matplotlib [23]

Bibliography

[1] Baseball Strike Zone Dimensions & Drawings | Dimensions.com.

[2] Major League Miscellaneous Year-by-Year Averages and Totals.

[3] Matplotlib — Visualization with Python.

[4] MLB Pitch Data 2015-2018.

[5] National League of baseball is founded - HISTORY.

[6] NumPy.

[7] pandas - Python Data Analysis Library.

[8] scikit-learn: machine learning in Python — scikit-learn 1.0.2 documentation.

[9] Strike Zone | Glossary.

[10] TrackMan Baseball V3 - Software.

[11] An Exploration of MLB Umpires' Strike Zones, May 2018.

[12] Robot Umpires Strike Zone: It's Not as Simple as You Think, January 2018.

[13] Rethinking the Strike Zone: It's Not a Square, February 2019.

[14] Student's t-distribution, April 2022. Page Version ID: 1083494571.

[15] Eric A. E. Gerber and Bruce A. Craig. A mixed effects multinomial logistic-normal model for forecasting baseball performance. *Journal of Quantitative Analysis in Sports*, 17(3):221–239, September 2021. Publisher: De Gruyter.

[16] David J. Hunter. New metrics for evaluating home plate umpire consistency and accuracy. *Journal of Quantitative Analysis in Sports*, 14(4):159–172, December 2018. Publisher: De Gruyter.

[17] Fränzi Korner-Nievergelt, Tobias Roth, Stefanie von Felten, Jérôme Guélat, Bettina Almasi, and Pius Korner-Nievergelt. Chapter 12 - Markov Chain Monte Carlo Simulation. In Fränzi Korner-Nievergelt, Tobias Roth, Stefanie von Felten, Jérôme Guélat, Bettina Almasi, and Pius Korner-Nievergelt, editors, *Bayesian Data Analysis in Ecology Using Linear Models with R, BUGS, and STAN*, pages 197–212. Academic Press, Boston, January 2015.

[18] Timothée Lesort, Natalia Díaz-Rodríguez, Jean-Franois Goudou, and David Filliat. State representation learning for control: An overview. *Neural Networks*, 108:379–392, December 2018.

[19] Brian M. Mills. Technological innovations in monitoring and evaluation: Evidence of performance impacts among Major League Baseball umpires. *Labour Economics*, 46:189–199, June 2017.

[20] Steve Nelson. Understanding the Strike Zone in Baseball.

[21] Adam Reiter. Major League Baseball: Why Games Need to Be Shortened and How It Can Be Done.

[22] Tom Verducci. The technology boom is fundamentally altering baseball.

[23] Michael Waskom. seaborn: statistical data visualization. *Journal of Open Source Software*, 6(60):3021, April 2021.