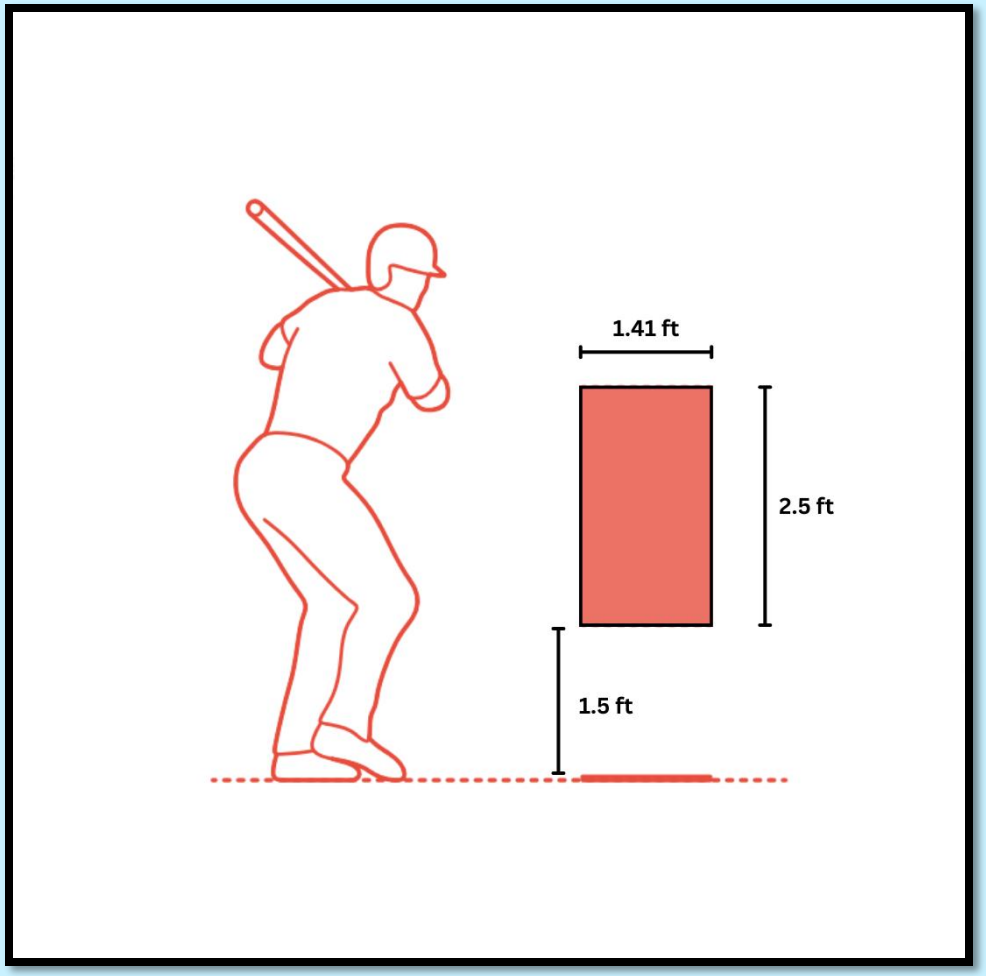
Robot Umpires: The necessity to take human error out of modern-day sports officiating

Grant Melvin & Jaxon Bauer

Introduction



- Motivation

  Human error plagues modern sports officiating, especially in the case of Major League Baseball (MLB), where umpires are tasked with calling every pitch thrown as a strike or a ball depending on their own perception of an imaginary strike zone. We want to put a microscope on the data and find the most common issues with this problem, and underline why robotic umpires could be a huge benefit for modern-day sports.

- Context
  - Accuracy of human officiating remains a pressing concern
  - MLB Umpires frequently make incorrect calls, which can have major impacts on the outcome of a game.

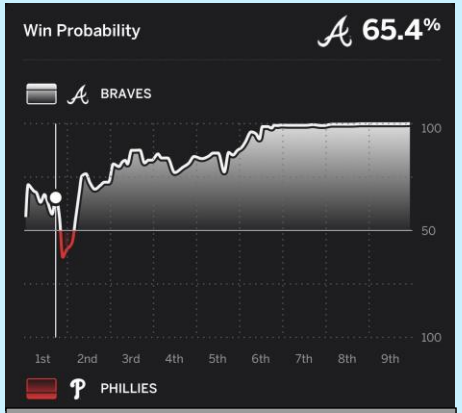
- Problem Statement
  - Despite the dedication and expertise of human officials in modern sports, the frequency of erroneous decisions remains unacceptably high, highlighting the urgent need for a technologically-driven approach to enhance accuracy and restore trust in the officiating process.

Max Fried Missed Strike Call

Importance of Situation

- The umpire calls the pitch a ball.
- The batter eventually gets walked, resulting in a run scored.
- Another batter gets a hit, resulting in two more runs.
- Braves end up winning 12-4

The situation: score 2-0 Braves, runners on all three bases, 2 outs, 2-2 count



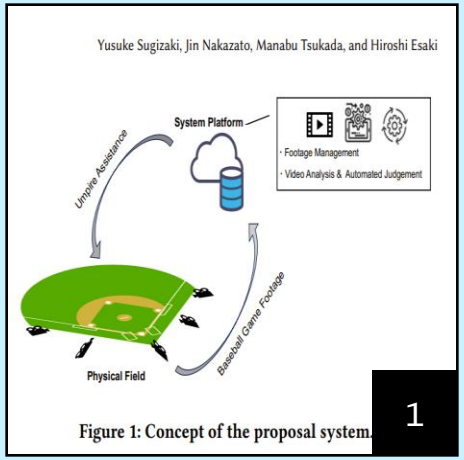
The win probability *before* umpire error

The win probability *after* umpire error

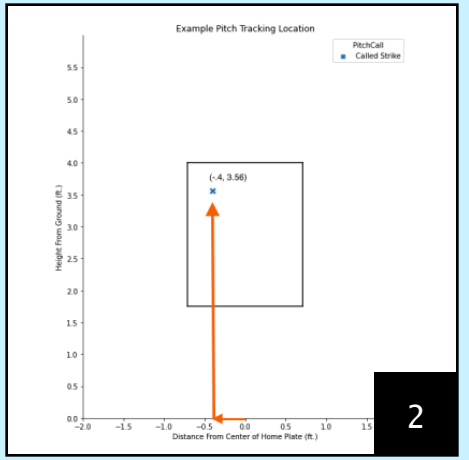
Related Works

Works that discuss human error in sports and propose machine-driven solutions.

1. Umpire Assistance System in Baseball Game
2. Estimating the Impact of Automated Umpiring in Baseball via Monte Carlo Simulation
3. DRS Assisting Umpire Using Computer Vision



1



2

3

The Data

"Statcast Era" Pitcher Dataset

- Pitcher data was harvested from Baseball Savant
  - Top 50 pitchers by pitch count since 2017
  - Over 500,000 pitches gathered
  - Filtered to include only pitching metrics
- Dataset created for measuring individual performance of players by position
- Official Dataset of Major League Baseball (MLB)

Cleaning the Dataset

- Original dataset includes both pitching and hitting metrics
  - Set filters to only include pitcher data
  - Determined the values that were beneficial from pitcher data
  - Performed further filtration to only gather what fields were necessary

Example of Cleaned Data:


```
_id: ObjectId('660c4cb942a669b002c71f0e')
player_name : "Nola, Aaron"
Pitcher_Throwing_arm : "R"
game_date : "9/26/2023"
count : "0-0"
pitch_name : "Sinker"
pitch_type : "SI"
release_speed : "89.6"
release_spin_rate : "2206"
horizontal_pitch_movement : "-1.5"
vertical_pitch_movement : "0.95"
description : "ball"
zone : "14"
plate_location_horizontal : "1.02"
plate_location_vertical : "1.33"
```

Methodology

Loading Data to MongoDB

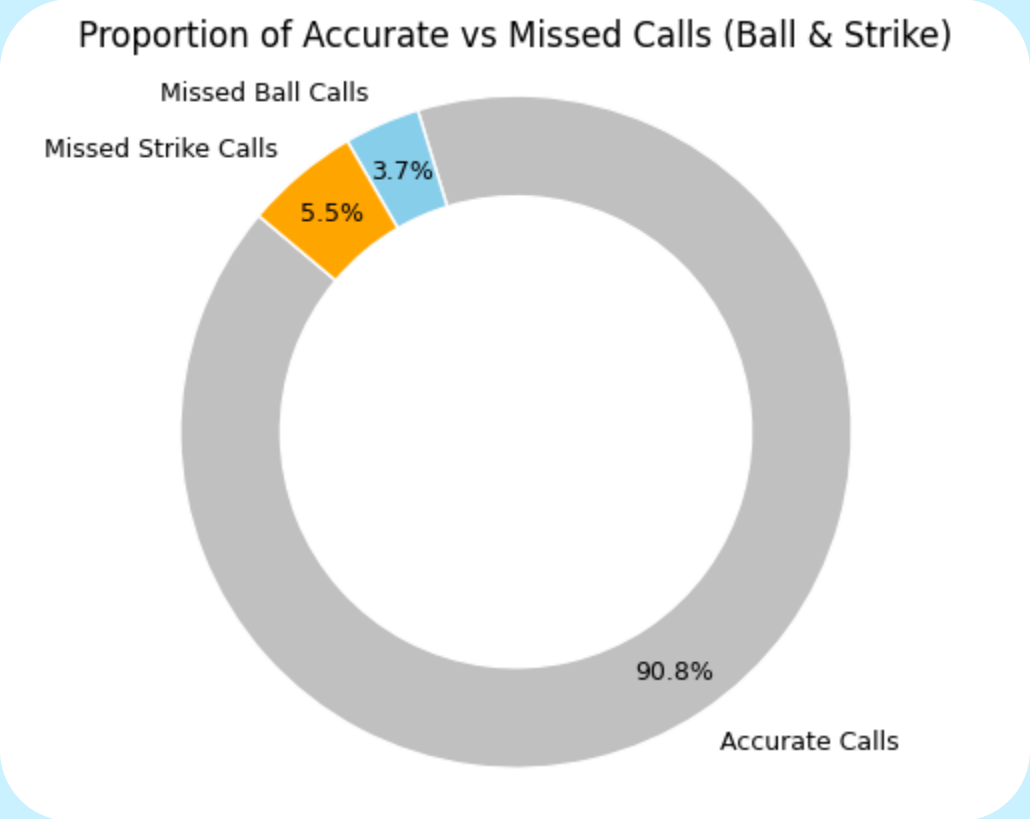
```
// Changes CSV -> JSON and pushes it into tempData
await csvtojson()
  .fromFile(filePath)
  .then((source) => {
    for (let i = 0; i < source.length; i++) {
      const oneRow = {
        player_name: source[i]["player_name"],
        Pitcher_Throwing_arm: source[i]["p_throws"],
        game_date: source[i]["game_date"],
        count: source[i]["balls"] + "-" + source[i]["strikes"],
        pitch_name: source[i]["pitch_name"],
        pitch_type: source[i]["pitch_type"],
        release_speed: source[i]["release_speed"],
        release_spin_rate: source[i]["release_spin_rate"],
        horizontal_pitch_movement: source[i]["pfx_x"],
        vertical_pitch_movement: source[i]["pfx_z"],
        description: source[i]["description"],
        zone: source[i]["zone"],
        plate_location_horizontal: source[i]["plate_x"],
        plate_location_vertical: source[i]["plate_z"],
      };
      tempData.push(oneRow);
    }
  });
```

Methodology

Visualizing the Data with MatPlotLib

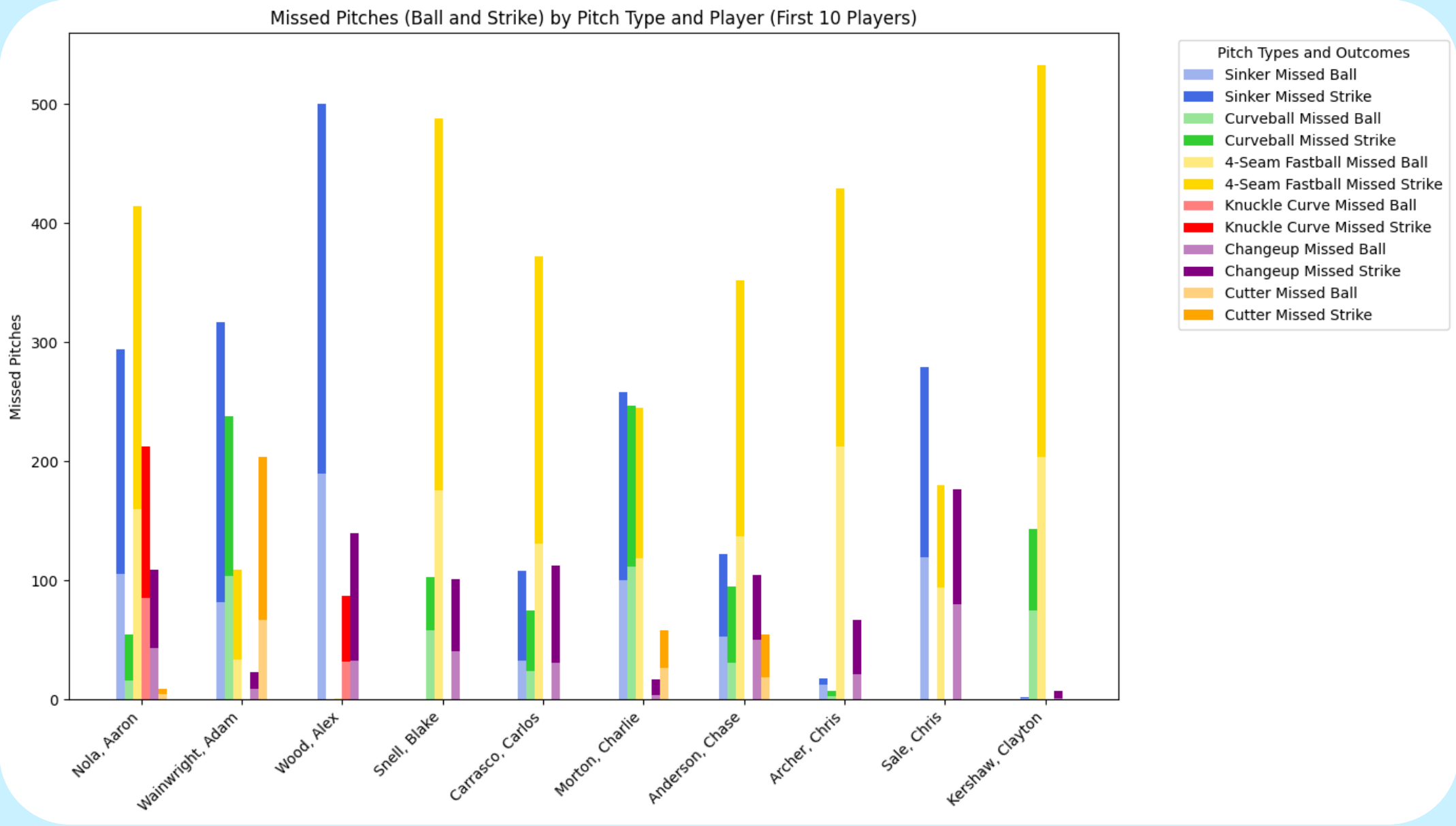
```python
# Setup for plot
fig, ax = plt.subplots(figsize=(14, 8))

# Parameters for the bars
width = 0.15  # Width of each bar to fit all pitch types for each player within their cluster

# Calculate the spacing between clusters to position them correctly
cluster_spacing = width * 2
# Number of players to display
num_players = min(10, len(player_data))  # Ensure we don't exceed the actual number of players
# Base x positions for each player's cluster of pitch types, adjusted for the number of players
x_base = np.arange(num_players) * (len(pitch_colors) + 1) * cluster_spacing

# Creating a new dictionary for the first 10 players to get their initials
first_10_players_initials = {player_name: initials for player_name, initials in islice(player_initials_dict.items(), num_players)}

# Loop over each player to create their cluster of pitch types, limited to the first 10 players
for player_idx, (player_name, player_stats) in enumerate(islice(player_data.items(), num_players)):
    # Offset within the cluster for the current pitch type
    offset = 0
    for pitch_type, color in pitch_colors.items():
        pitch_data = player_stats['pitch_types'].get(pitch_type, {})
        total = pitch_data.get('Total', 0)
        missed = pitch_data.get('Missed', 0)
        non_missed = total - missed  # Calculate the non-missed portion

        # Position for this pitch type within the cluster
        pos = x_base[player_idx] + offset

        # Plot missed pitches at the bottom
        ax.bar(pos, missed, width, color=color, alpha=0.5, label=f'{pitch_type} Missed' if player_idx == 0 else "")
        # Stack the non-missed portion on top to show total pitches
        ax.bar(pos, non_missed, width, bottom=missed, color=color, label=f'{pitch_type} Total' if player_idx == 0 else "")

        # Move offset for the next pitch type within the cluster
        offset += cluster_spacing

# Customizing the chart
ax.set_ylabel('Number of Pitches')
ax.set_title('Pitch Types by Player: Total and Missed Pitches (First 10 Players)')
# Adjust x-ticks to center them for each player's cluster of pitch types
ax.set_xticks(x_base + (len(pitch_colors) * cluster_spacing) / 2 - cluster_spacing)
# Setting x labels to be the initials from the first 10 players
ax.set_xticklabels([initials for initials in first_10_players_initials.values()])

# Simplifying the legend to only show once per pitch type
handles, labels = ax.get_legend_handles_labels()
unique_labels = dict(zip(labels, handles))
ax.legend(unique_labels.values(), unique_labels.keys(), title="Pitch Types and Outcomes", bbox_to_anchor=(1.05, 1), loc='upper left')

plt.tight_layout()
plt.show()
```

Results

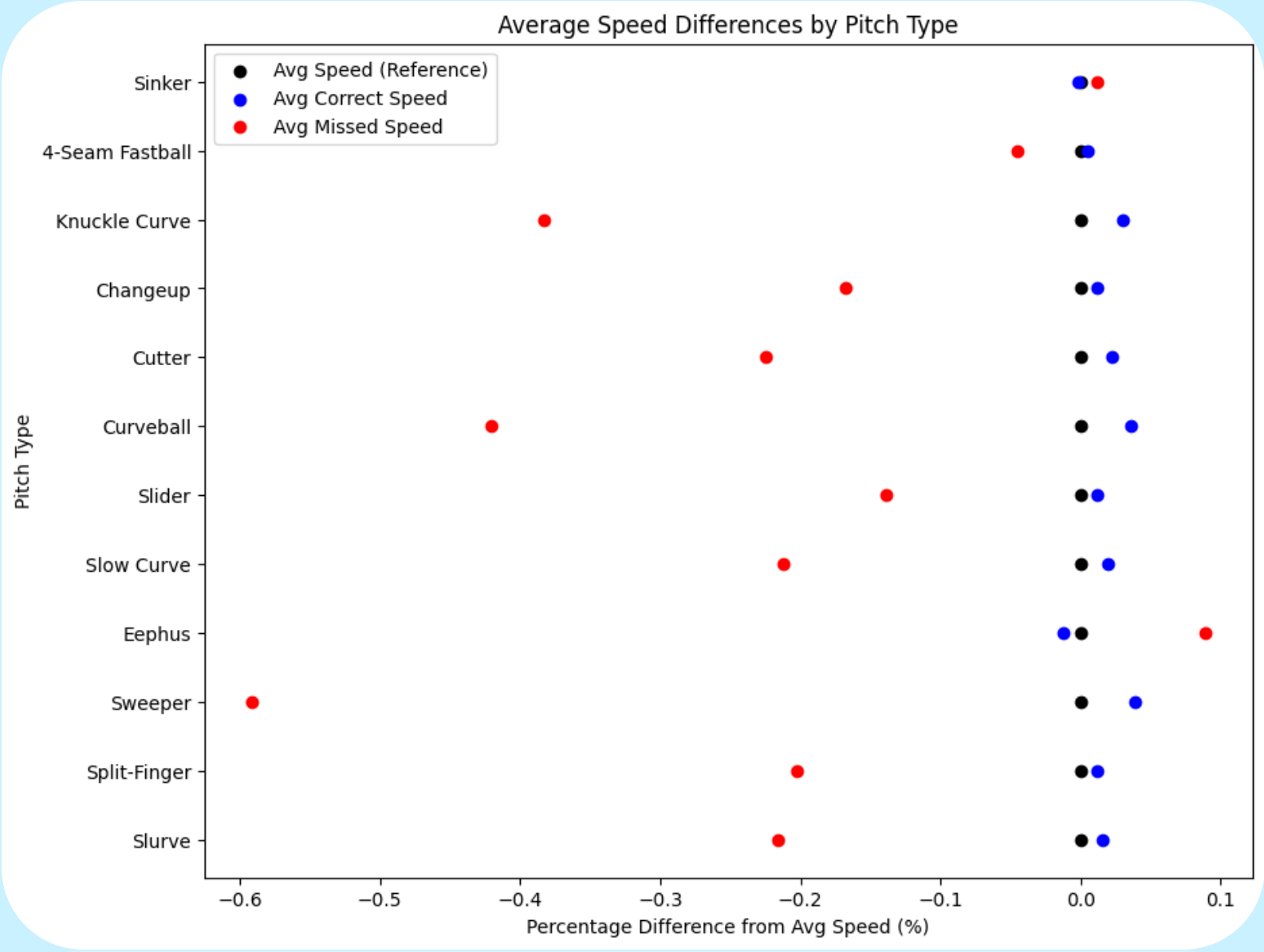


10% error rate is the magic number

Results

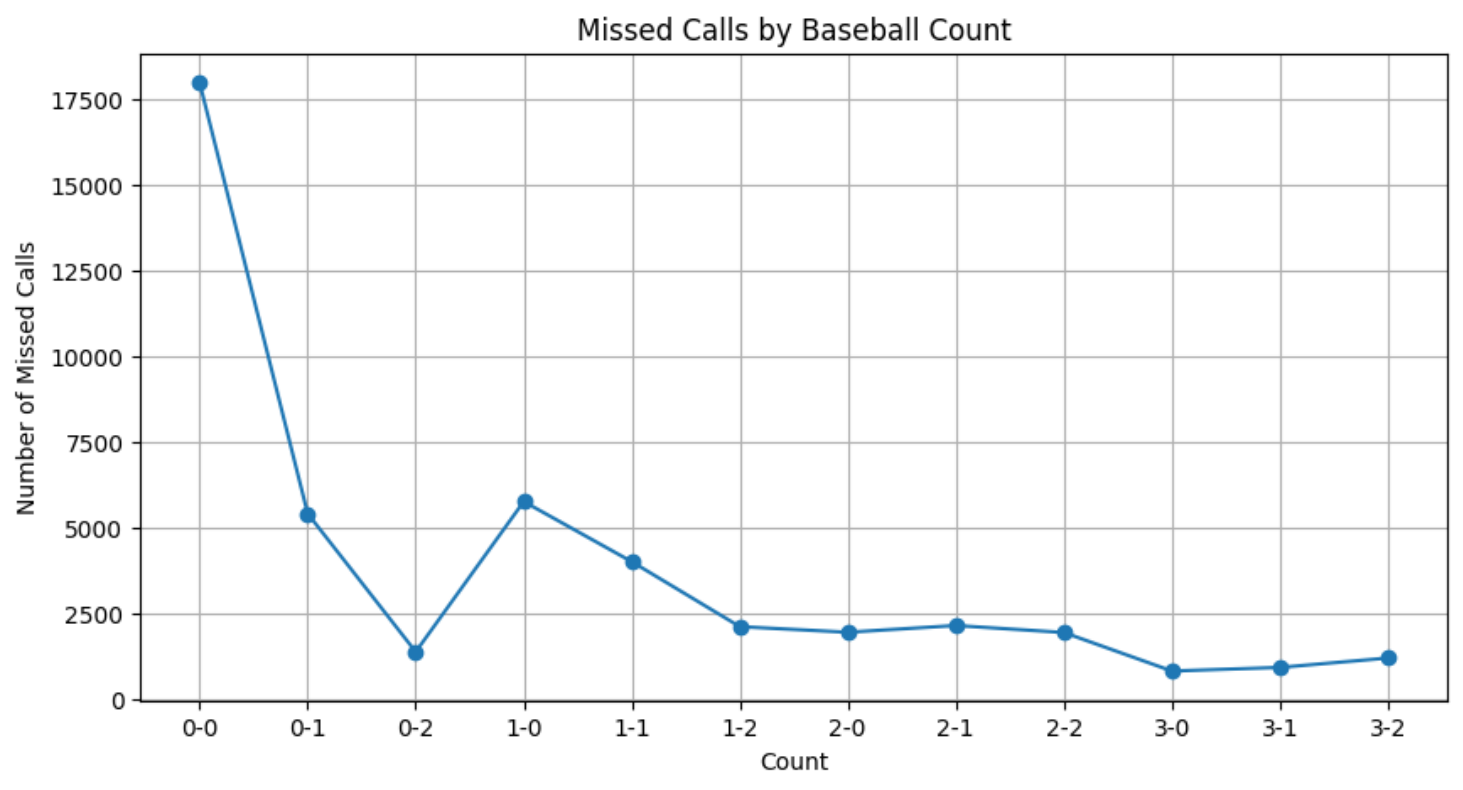

Missed Pitches (Ball and Strike) by Pitch Type and Player (First 10 Players)
Missed Pitches
0
100
200
300
400
500
Nola, Aaron
Wainwright, Adam
Wood, Alex
Snell, Blake
Carrasco, Carlos
Morton, Charlie
Anderson, Chase
Archer, Chris
Sale, Chris
Kershaw, Clayton
Pitch Types and Outcomes
Sinker Missed Ball
Sinker Missed Strike
Curveball Missed Ball
Curveball Missed Strike
4-Seam Fastball Missed Ball
4-Seam Fastball Missed Strike
Knuckle Curve Missed Ball
Knuckle Curve Missed Strike
Changeup Missed Ball
Changeup Missed Strike
Cutter Missed Ball
Cutter Missed Strike

Results



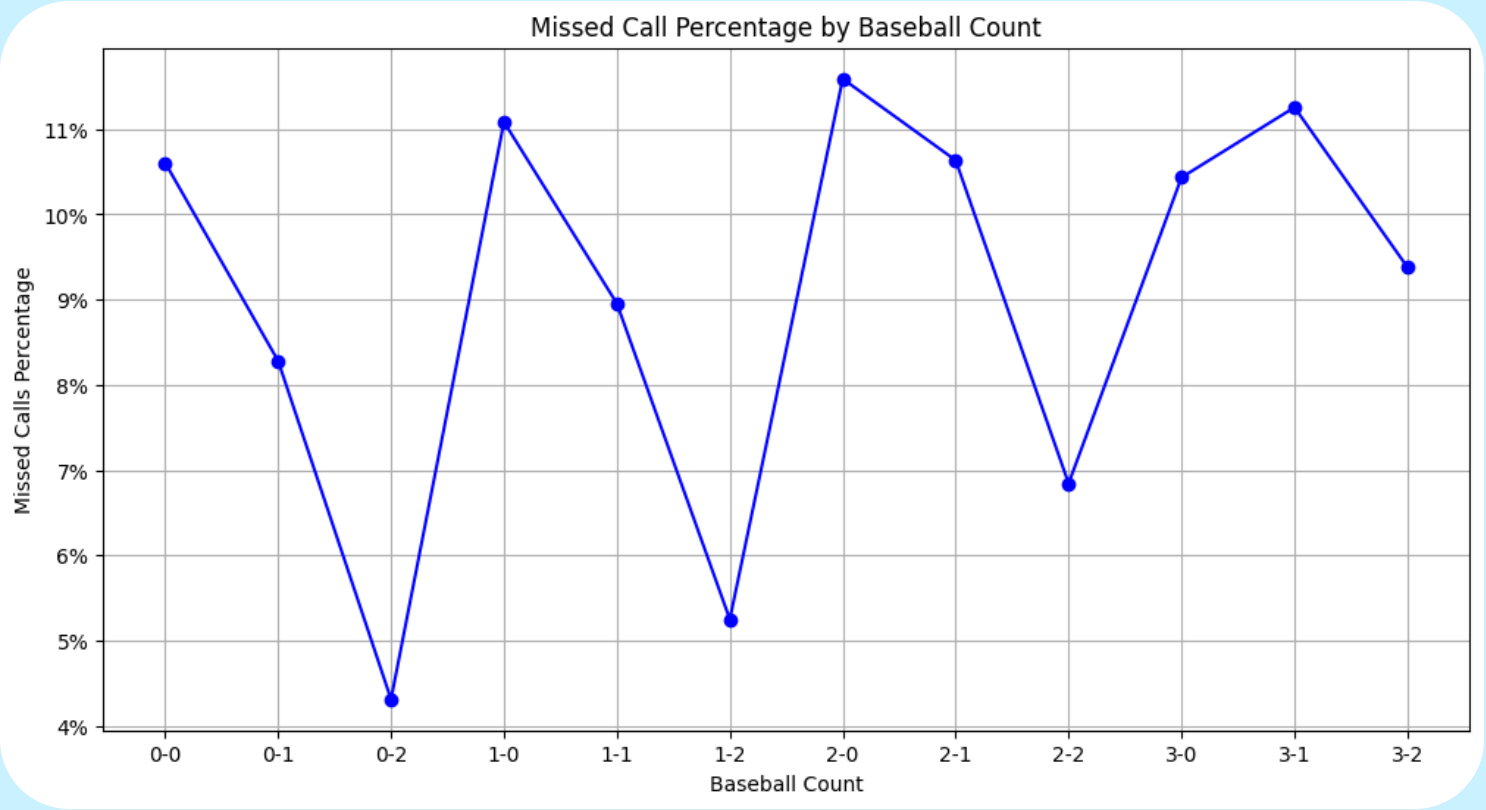
Speed analysis does not align with hypothesis going in

Results



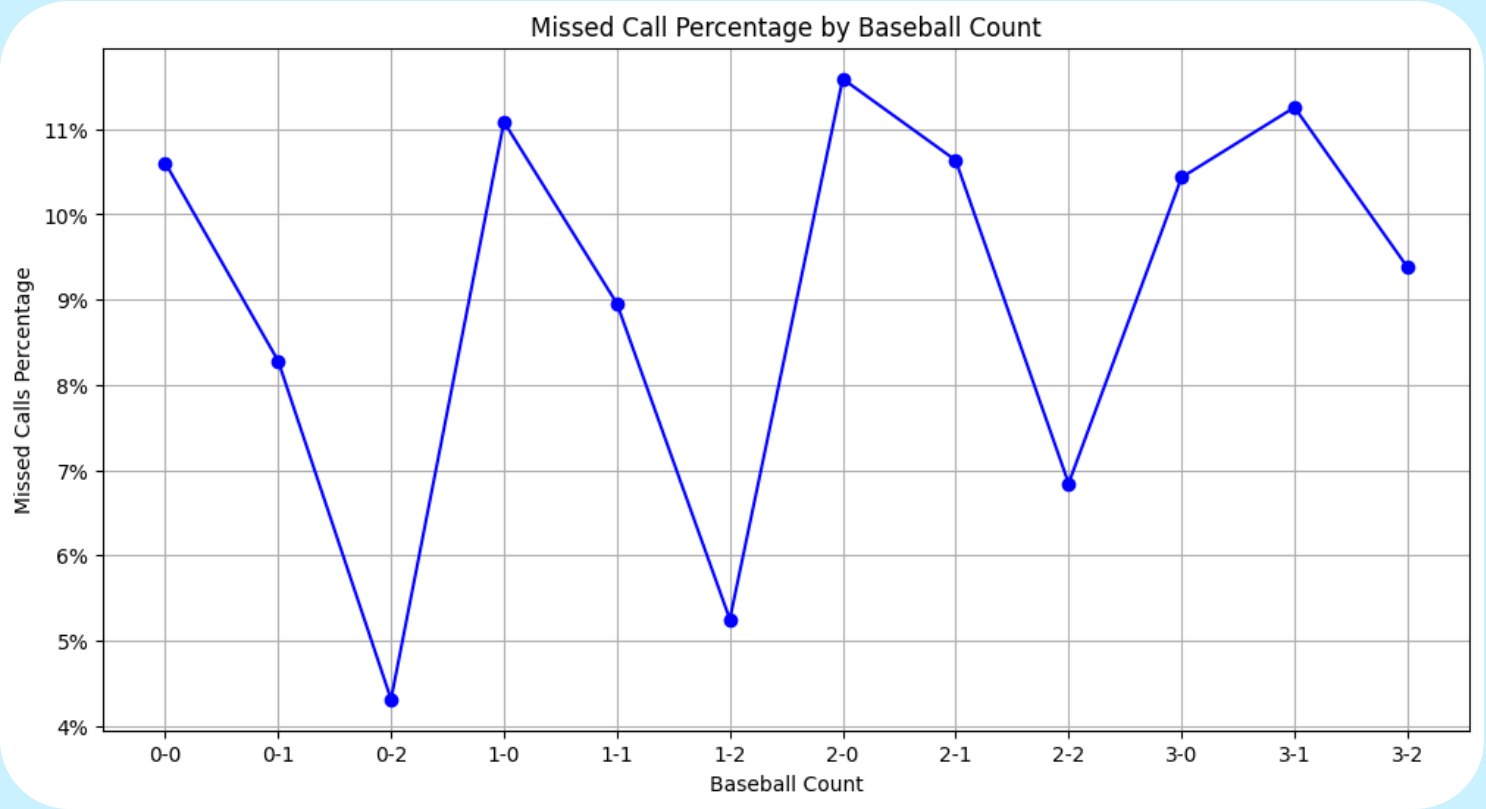
Appears to be heavily skewed towards error occurring on the 0-0 count as opposed to everything else. Why?

Results



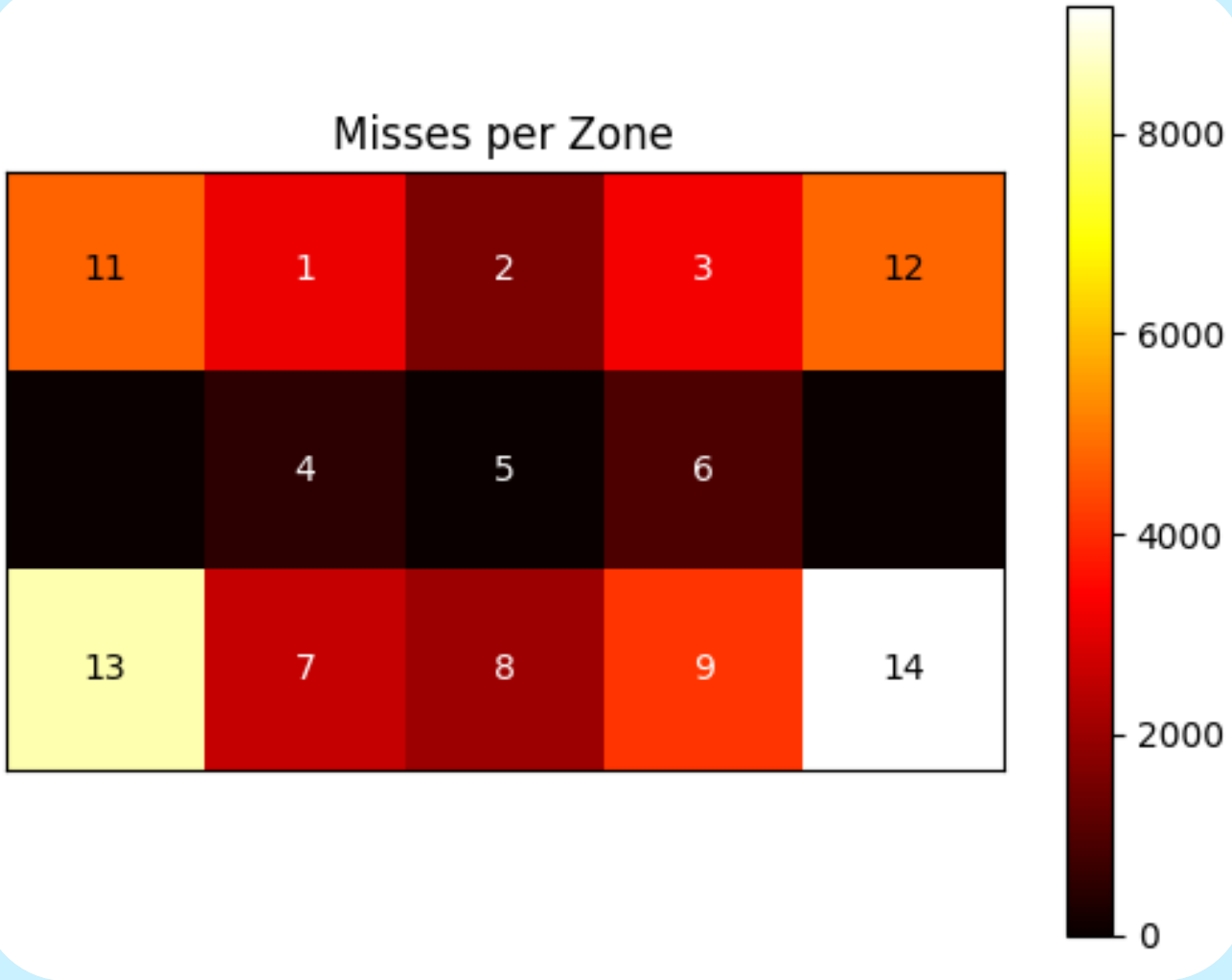
Ratios instead of flat numbers

Results



Ratios instead of flat numbers

Results



More likely to be called an incorrect strike outside of the zone than a ball inside of the zone


Ba
Lu
lawrencium
103
Lr
rutherfordium
104
Rf
tantalum
73
Ta
dubnium
105
Db
tungsten
74
W
seaborgium
106
Sg
Mo
Tc
rhenium
75
Re
bohrium
107
Bh
Cr
Fe
Nd
Pm

Thank You

Grant Melvin & Jaxon Bauer