



UNIVERSITÀ
DI TRENTO

Università Degli Studi di Trento
Artificial Intelligence Systems

Extracting Significant Features in Atari Games

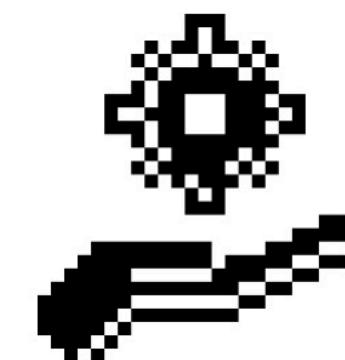
Signal, Image and Video Project

Partner di Progetto

Sorrentino Francesco

Richichi Andrea

Introduction



Atari games, with their simplicity and strategic depth, are ideal for testing AI through visual data processing.



- Used Arcade Learning Environment (ALE) for data collection.
- Implemented a pipeline for frame acquisition and processing.
- Applied techniques like SIFT, segmentation, and motion tracking.



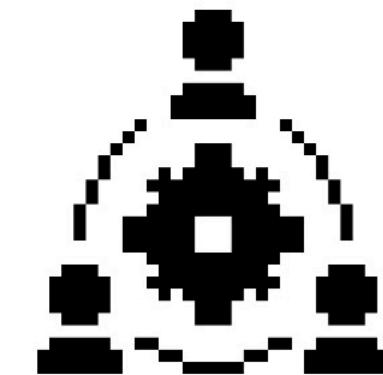
Extract visual features from Atari games to support AI analysis with a structured data pipeline.

Why Use Atari Games?

Atari games are perfect for testing image processing thanks to their simplicity, iconic design, and clear challenges. Their low resolution and defined trajectories offer a controlled environment for algorithm evaluation and symbolic abstraction.



Simplicity



**Iconic and
Manageable**



Game Choices

Arcade Learning Environment (ALE)

- **AI Research Framework**

Built on the Stella Atari 2600 emulator, ALE provides accurate emulation of over 900 games for single and multi-agent studies.

- **Data Sources**

Provides RAM state and pixel frames; this project uses frames for analysis.

- **Project Focus**

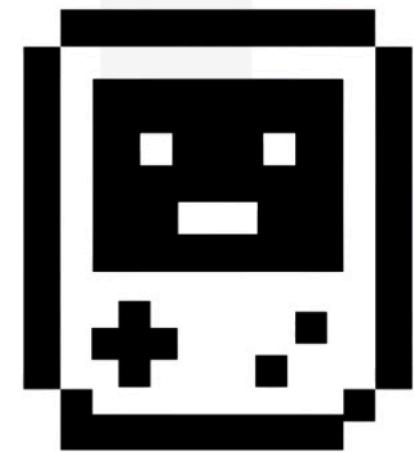
Studies Breakout and Skiing using ALE's Python interface.



Acquisition Process

To analyze gameplay, we collect frames from two games: Breakout & Skiing.

We implemented two types of acquisition methods for this purpose:



Random Agent

Fast, automated gameplay
with lower scores.



Manual Player

Slower, human-driven
gameplay with higher scores.

BREAKOUT



SKIING



Attacked Problem

The project extracts structured data from frames to bridge the gap between raw pixels and meaningful AI information.

Extracted Data

From each frame, we extract key information to analyze objects and their behavior:

- **Scale-Invariant Feature Transform (SIFT)**
Identify stable features for object detection and pattern recognition.
- **Connected components information**
Determine object properties such as position, size, and shape.
- **Pixel lists**
Precisely locate objects by storing the coordinates of their pixels.
- **Partial distances**
Track movement over time to study object dynamics and interactions.

Methodologies

A structured process extracts key features, classifies objects, and tracks motion. The next slides illustrate the six essential steps.

1. Load Frames

A procedure loads frames in **BGR** and **Grayscale** to optimize processing:

- **Function:**

```
load_frames(game, step=100) return imgs, gray_imgs.  
  
#game – Specifies the game.  
#step – Skips frames (default: 100) to reduce redundancy.
```

- **Why?**

- **Skipping frames** speeds up processing and reduces similar images.
- **BGR & Grayscale** allow flexibility in feature extraction.

2. Find SIFT Key Points

The SIFT algorithm extracts key features with **scale and rotation invariance** through four steps:

- **Scale-Space Extrema Detection**

Detects keypoints using **Difference of Gaussians** (DoG) for efficient blob detection.

- **Keypoint Localization**

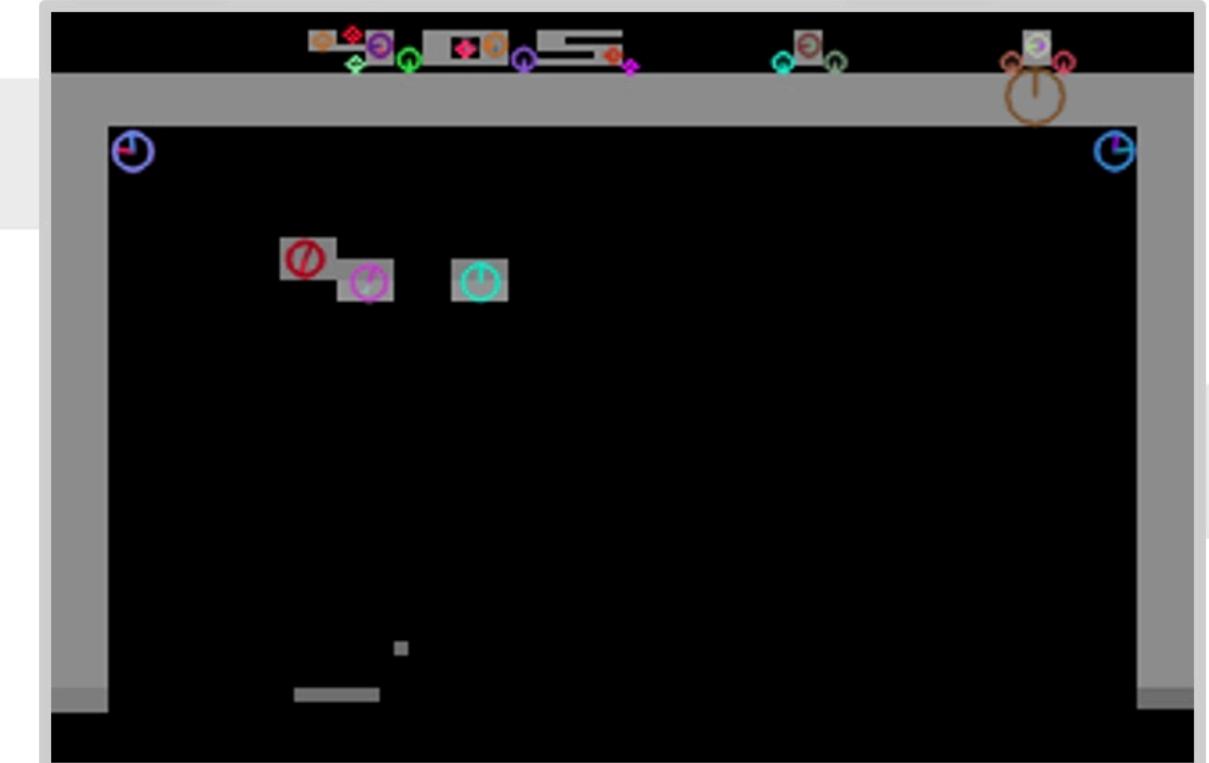
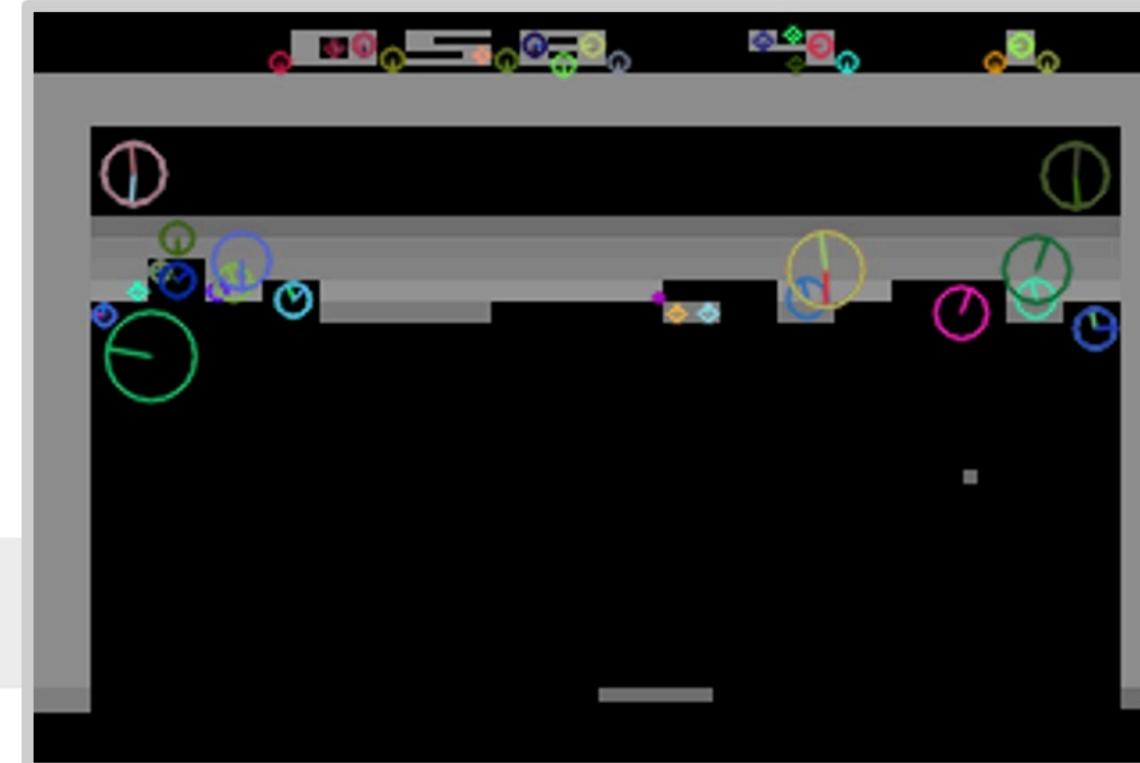
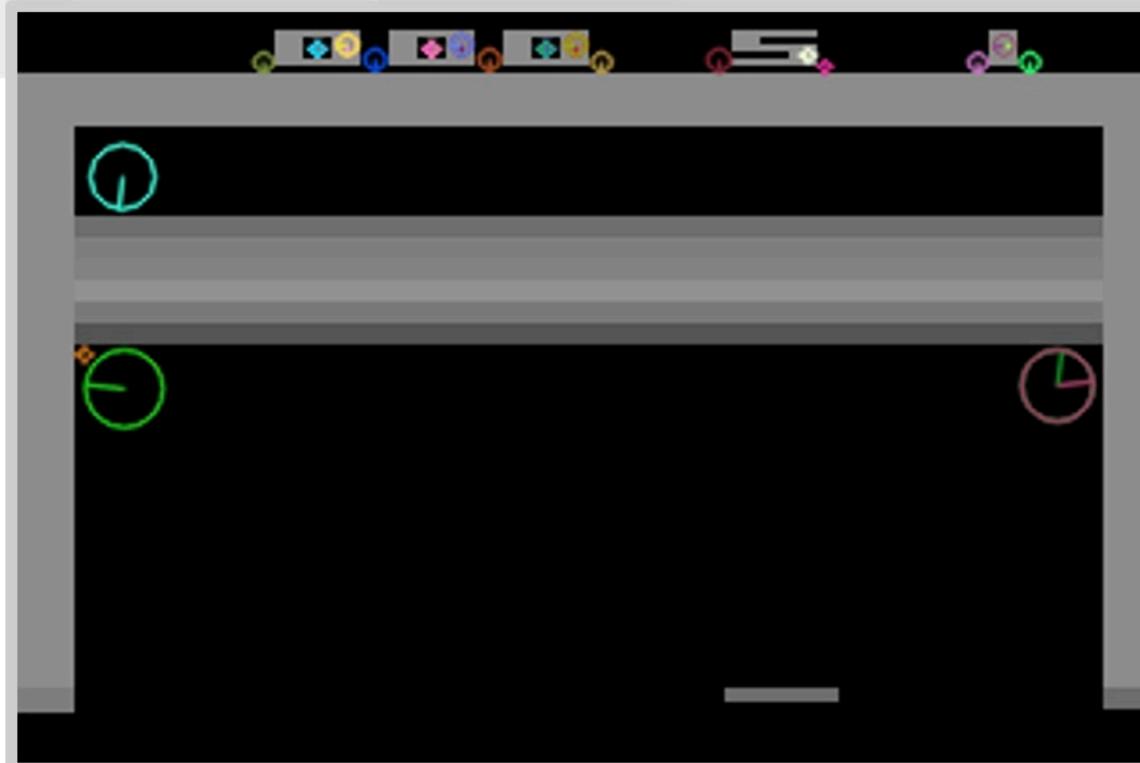
Filters out unstable keypoints using a **Taylor expansion** and contrast threshold.

- **Orientation Assignment**

Computes gradient histograms to make keypoints **rotation-invariant**.

- **Keypoint Descriptor**

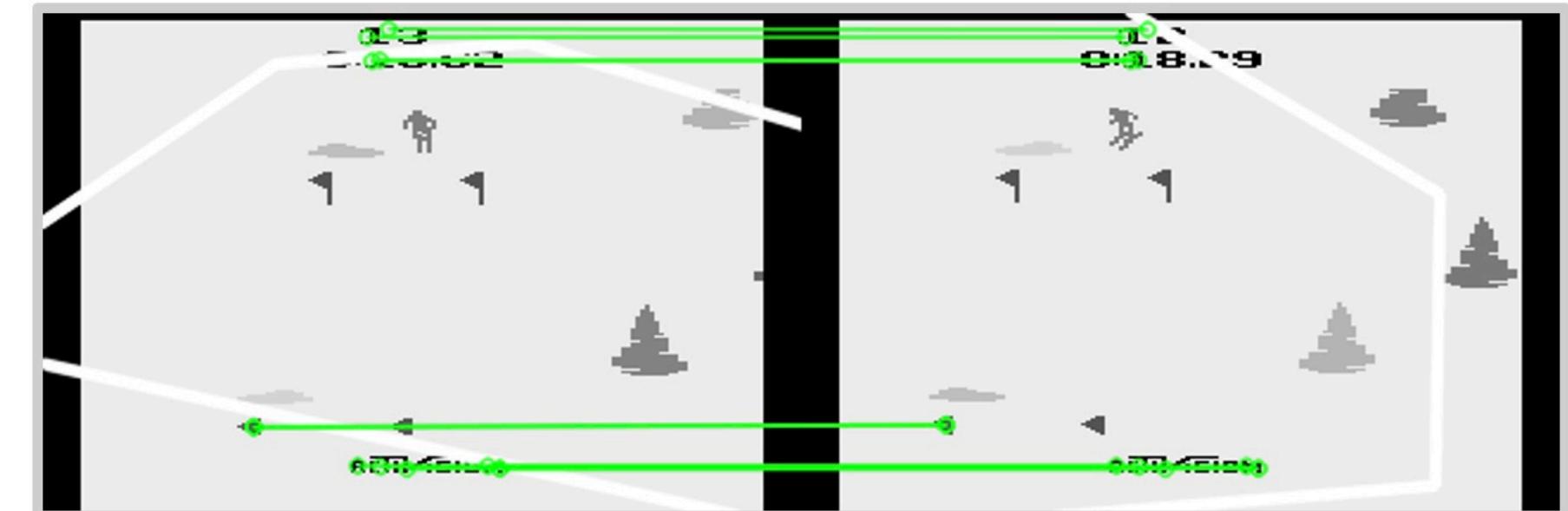
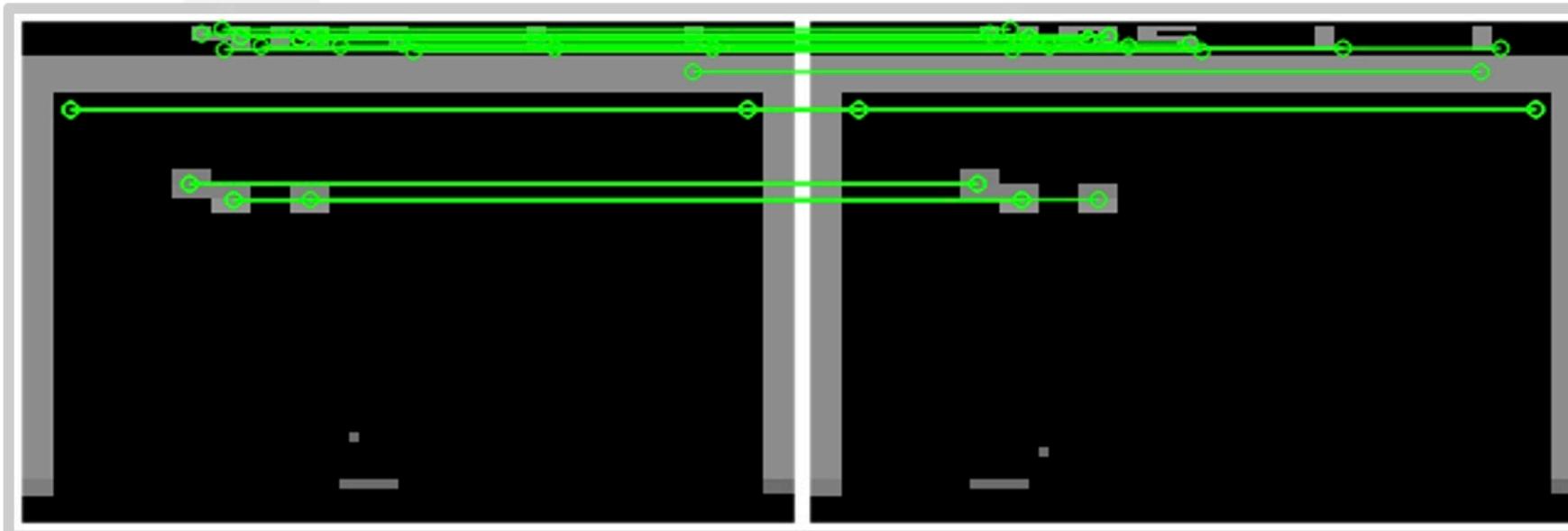
Generates a **128-value feature vector** for robust matching.



3. Feature Mapping

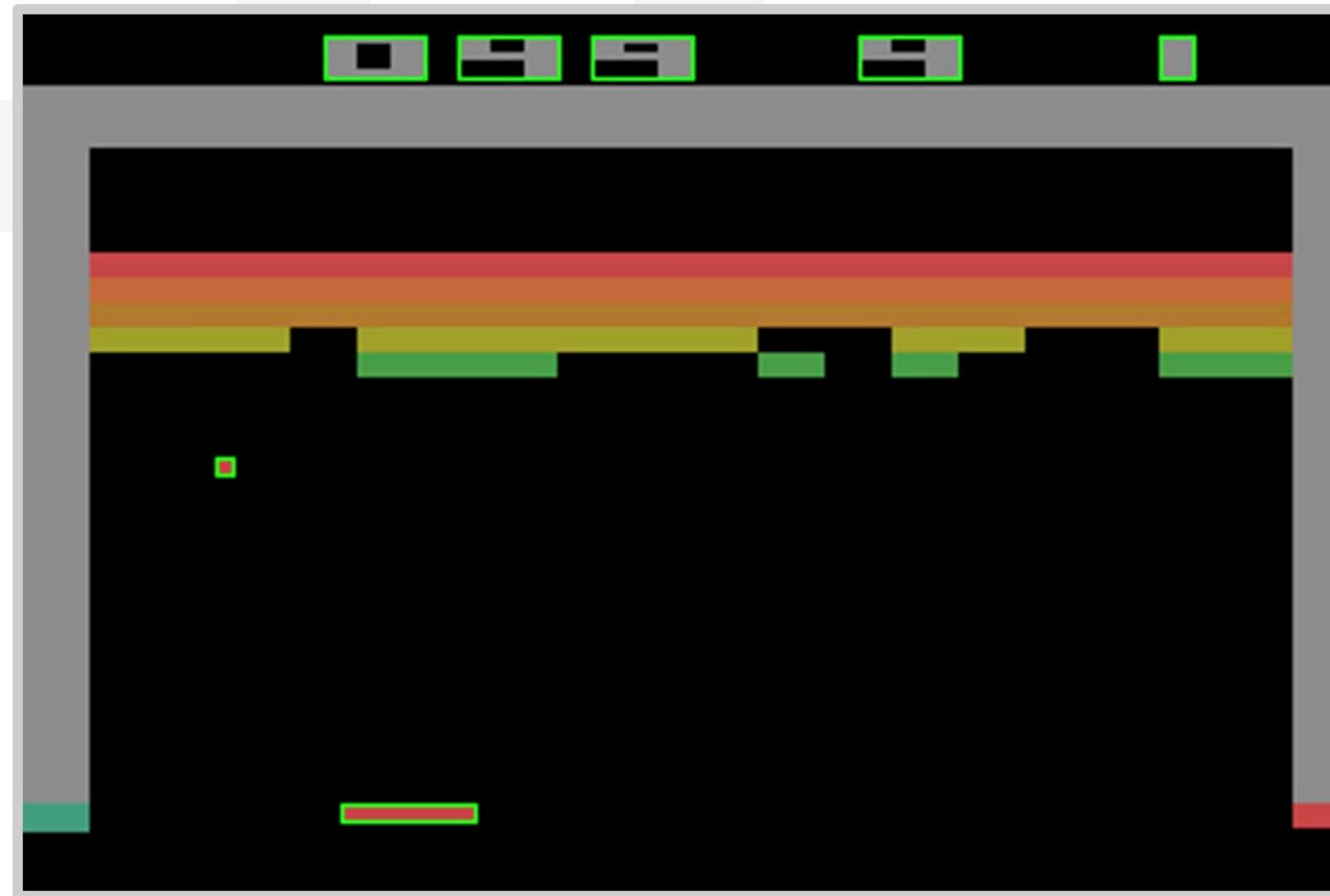
Once keypoints are detected in each frame, we **match** them between consecutive images:

- **Nearest Neighbor:** Finds the **two best matches** for each keypoint.
- **Lowe's Ratio Test:** Filters out **weak matches** for better reliability.
- **Homography:** Aligns images when enough matches are found.



4. Detect connected components

Objects in each frame are detected using **thresholding** and **component labeling**. Large background areas are filtered out, keeping only relevant objects.

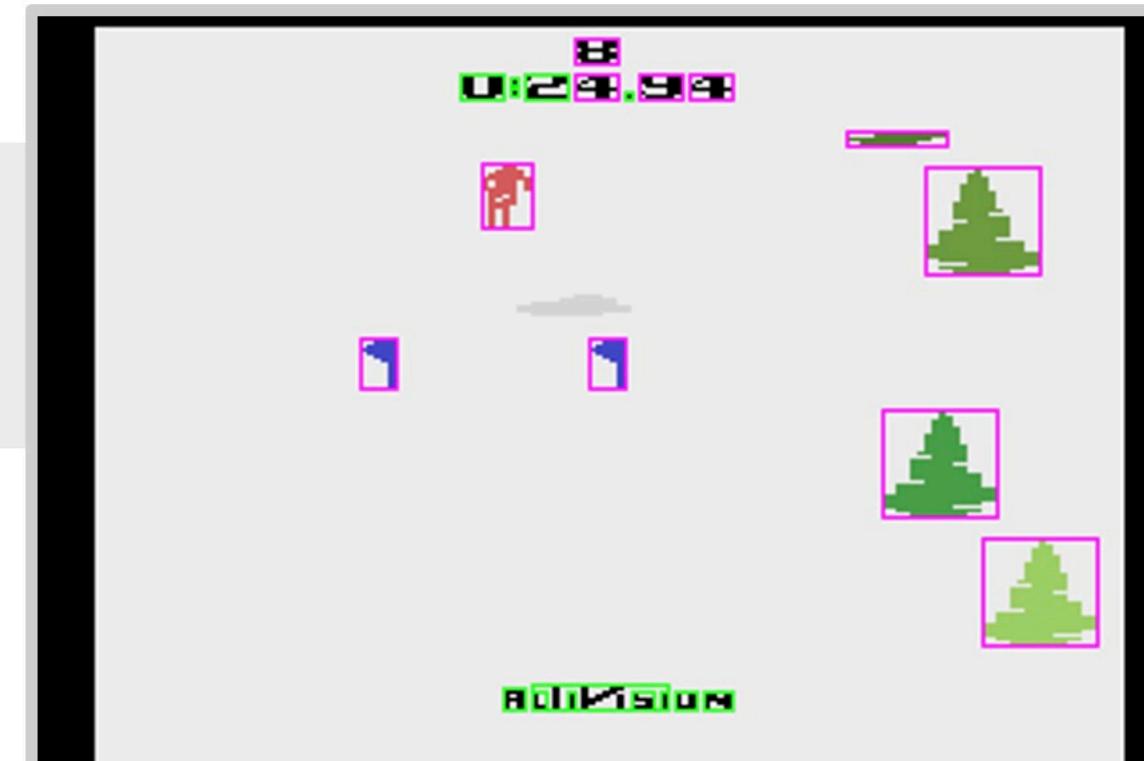


5. Object Classification

Objects are **static** if they retain pixel overlap across frames; otherwise, they are **mobile**. A threshold ensures accuracy.

- Function:

```
class_list ← [] # Let us define 0 as static and 1 as mobile
    for i in range(len(stats) - 1): # Iterate on every frame
        frame_class ← []
            for j in range(len(stats[i])): # Iterate on every object
                object_class ← 1 # Set class mobile
                    for n in range(len(stats[i + 1])): # Iterate on every object of successive frames
                        inter ← find_intersection(pixels_lists[i][j], pixels_lists[i + 1][n])
                        area ← stats[i][j, cv.CC_STAT_AREA]
                        ratio ← len(inter)/area
                        if ratio ≥ thresh: # Check ratio must be higher than a threshold value
                            object_class ← 0 # Set class still
                        frame_class.append(object_class)
                    class_list.append(frame_class)
    return class_list
```



● Static

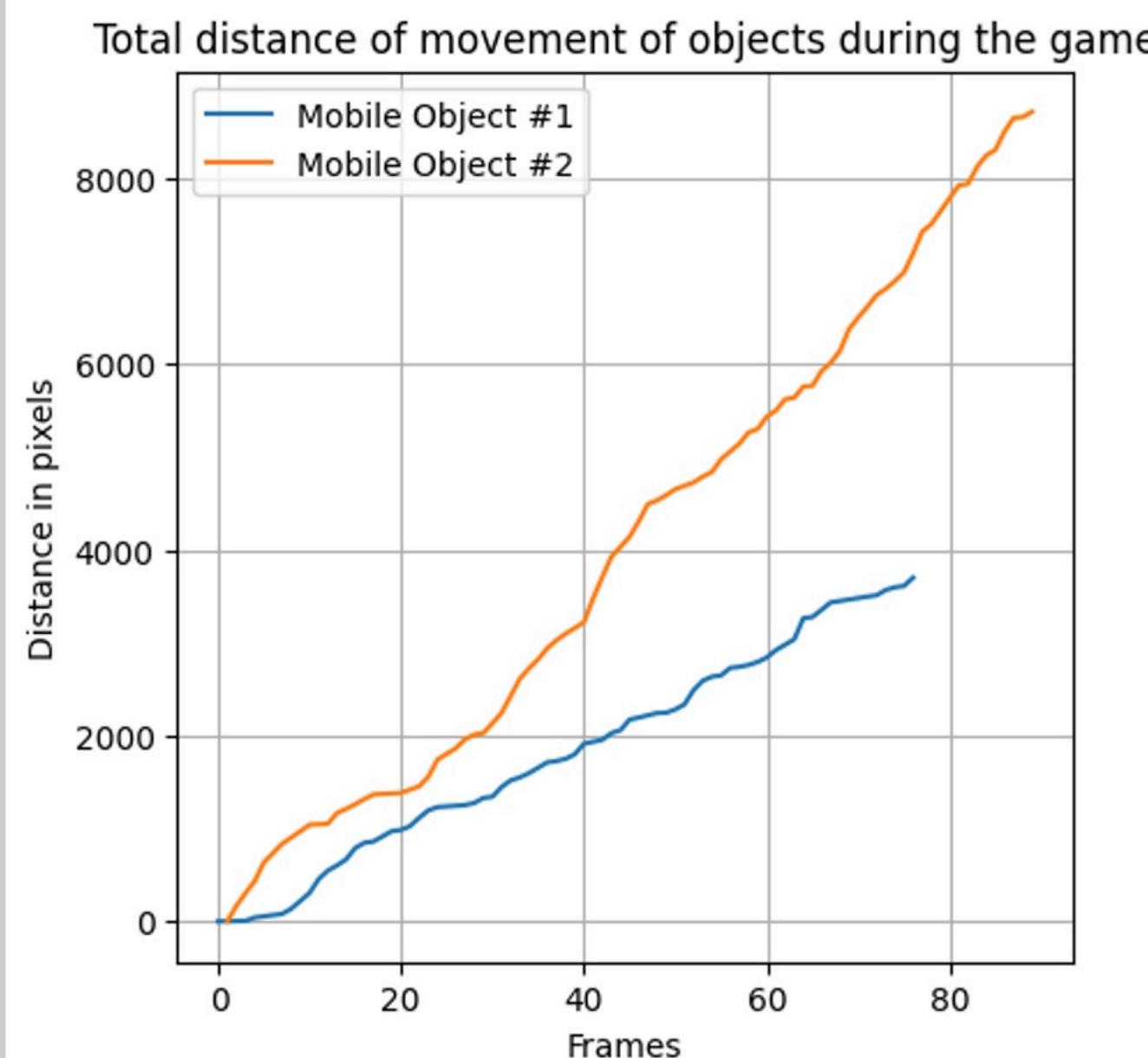
● Mobile

6. Object Tracking

The final step tracks **mobile objects** across frames and measures their movement over time. This helps in understanding object dynamics and interactions in the game.

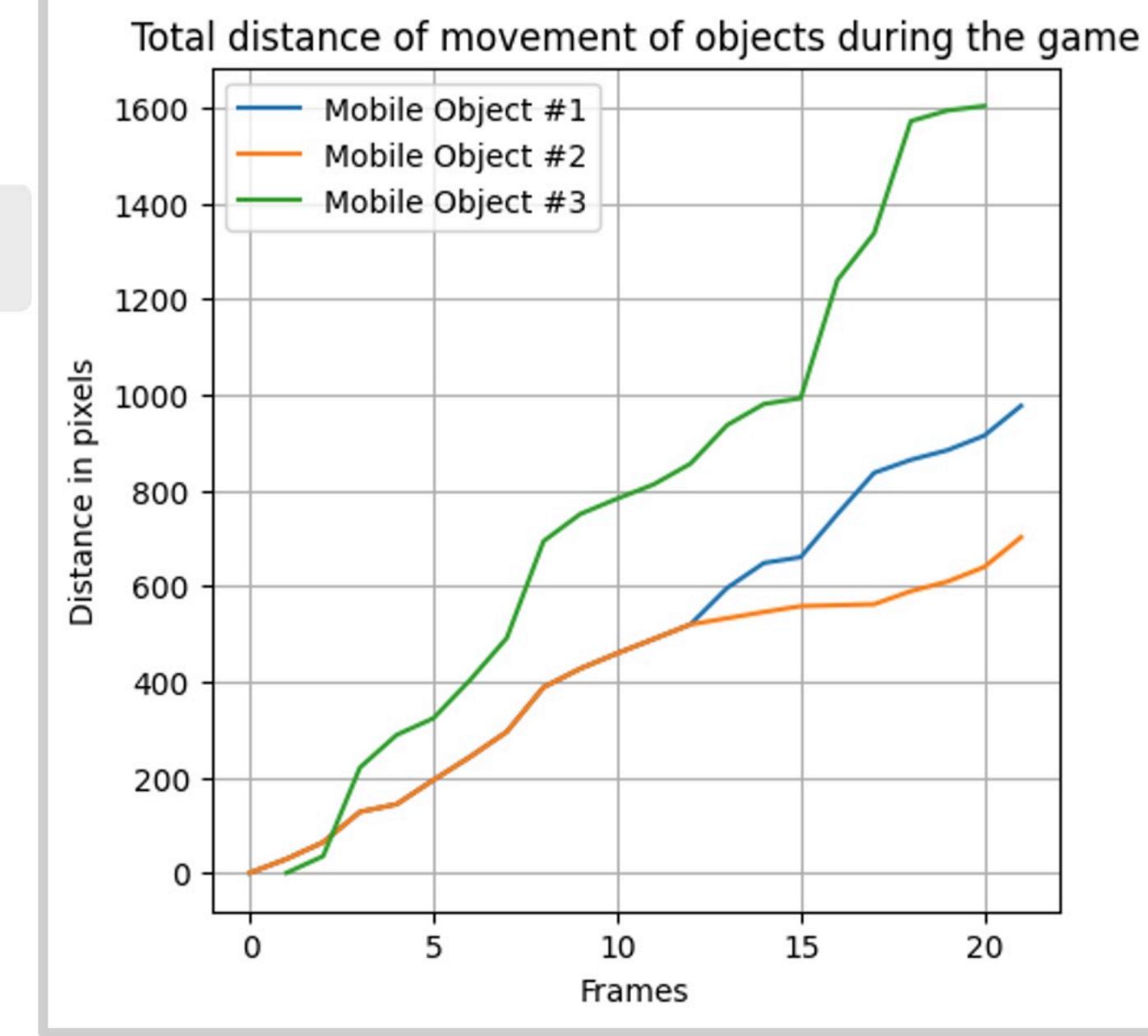
- **Objects are linked across frames** based on their position and size to ensure consistent tracking.
- **The graph visualizes total distance traveled** over time, helping to compare movement patterns.
- **Steeper curves indicate faster-moving objects**, showing which elements are the most dynamic.

BREAKOUT



- Paddle #1
- Ball #2

SKIING



- Right Flag #1
- Left Flag #2
- Player #3

Conclusion

The project successfully implemented the proposed method, demonstrating the value of **image processing in AI**.

Future work can explore **more games** and enhance feature extraction methods.

THANKS FOR YOUR
ATTENTION

Feel free to ask any questions