Brick Breaker hit detection

Filip Volarić

SW, FTN Novi Sad

Introduction

Brick Breaker is a video game, in which the player mush smash a wall of bricks by deflecting a bouncing ball with a paddle.

In this project the goal is to count how many times does ball hit left or right wall.

Data

First step for solution is to get videos of Brick Breaker game.

Data set which will be used contains 10 videos of different length.

Resolution of each video is 960 x 720.

In file res.txt we have correct count of hits for each video.

Implementation

First we need to load all the videos. Then for each video frame by frame extract positions of balls.

Unlike balls, walls have fixed position, so we can get line coordinates from only one frame.

Once we obtained positions of walls and balls, we can compare those positions and see if ball hits the wall or not.

Once video is finished we compare real value, that we get from res.txt with predicted value, that we counted, and use it to calculate mean absolute error.

Walls

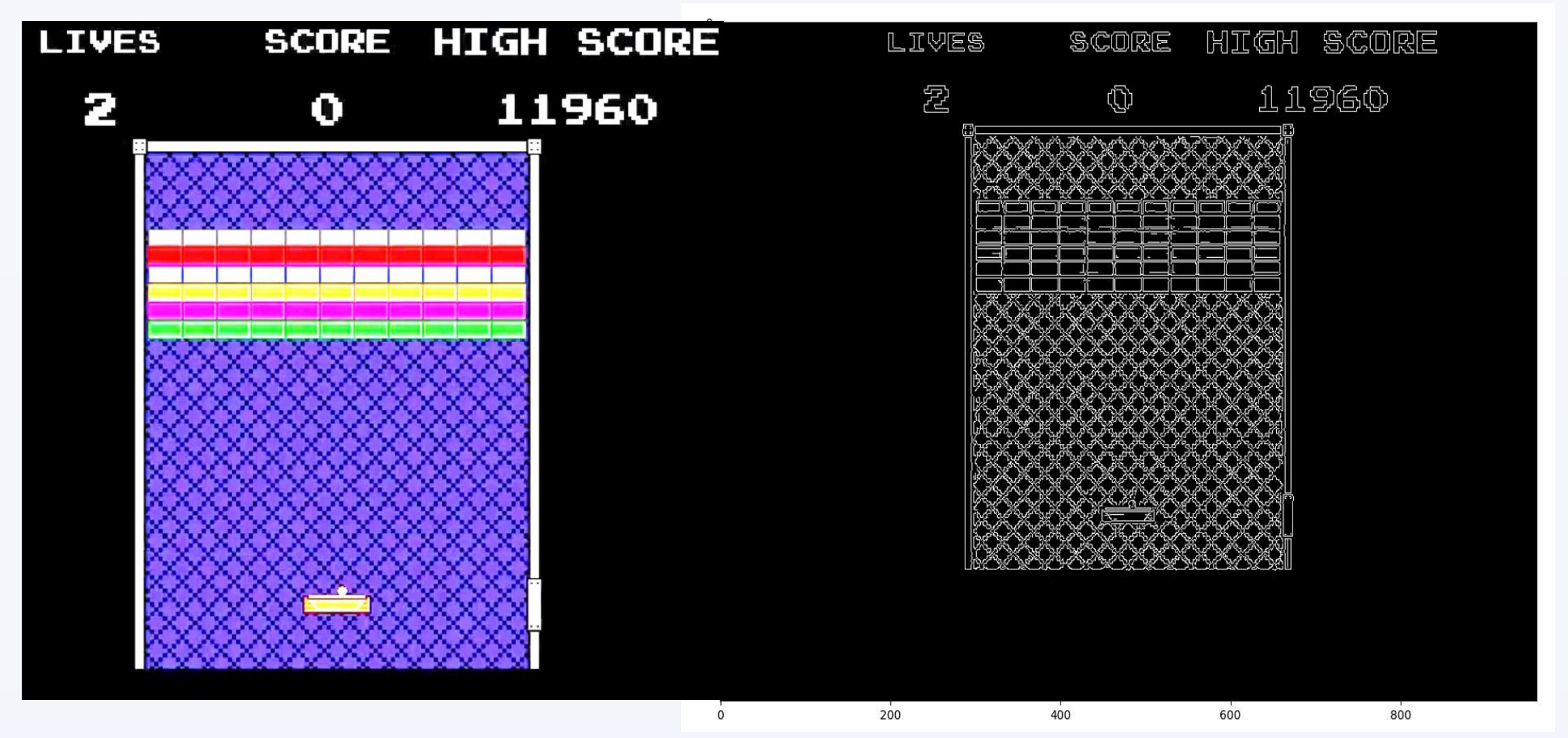
Extracting positions of left and right wall.

In first frame of each video we need to get coordinates for left and right wall.

1. Canny edge detector

The Canny edge detector is and edge detection operator that uses a multi-stage algorithm to detect a wide range of edges in images. It was developed by John F. Canny in 1986.

What we get as output of Canny method from opencv is library is binary image with edges only.



2. Hough transform

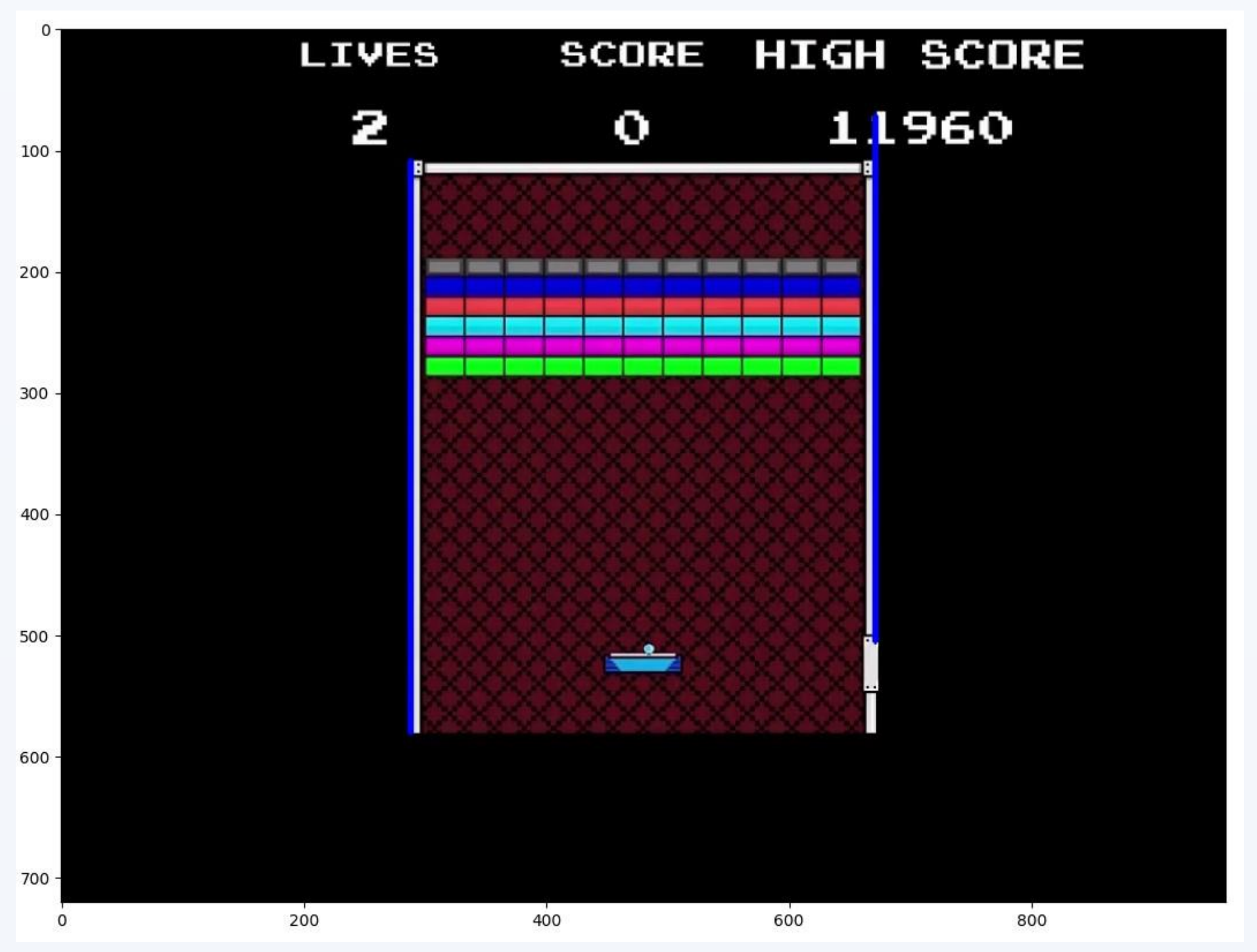
Hough transform is a feature extraction technique used in image analysis, computer vision, and digital image processing. The purpose is to find imperfect instances of objects within a certain class of shapes by a voting procedure.

We will use opencv implementation of probabilistic variant to the classical Hough transform: HoughLinesP.

With this method we are able to get all the lines that are longer than our desired limit.

As we know line is determined by two points, x1, y1 and x2, y2. In order to select only vertical lines, we must compare x1 and x2. If they are same, line is vertical.

Now that we have vertical lines it is time to select the ones that are walls. Left wall is the one on the far left and right wall is the on on the far right. So choosing the line with minimum X value will give us left wall, and choosing the one with maximum X value will give us right wall.



Balls

To extract balls from background we will use threshold. Thresholding is a process that creates a binary image based on setting a threshold value on the pixel intensity of the original image.

To do so, we must first convert colored frame to gray using opencv method.

Then we set threshold to 150 and from the binary image that we got, we must get positions of balls.

By using opencv method findContours we are able to get all contours from the binary image.

Balls are included in these contours. In order to get only balls from these contours, we must find a condition that works only for balls.

We can do that with method minEnclosingCircle and then selecting only contours which radius fits to a ball.

Now that we have our balls and walls positions, we can check for hits.

Hits

In each frame balls are in different position, so for each frame after getting their positions, we must check if ball hit the wall or not.

We do that by comparing X coordinates of balls center and wall. If the ball is close enough (closer than 20px) we count that as a hit. Since ball travels different distance for each frame, it can sometimes travel further or closer to the edge. By setting distance to px we are certain to see every hit but are risking to count one hit as two (ball coming towards the edge and ball bouncing off the edge).

We take care of this problem by not considering hits that are in two consecutive frames.

Results

To test if our program functions, we need to establish a validation metric. We will use MAE.

Mean absolute error (MAE) is a measure of errors between paired observations expressing the same prenomenon.

$$MAE = \frac{\sum_{i=1}^{n} |y_i - x_i|}{n}$$

Where y_i is the predicted value, and x_i is true value.

With these steps we managed to get a result of MAE = 0.3 which means our model only made 3 errors in the span of 10 videos.