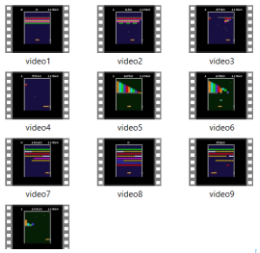
Brick Breaker Collision Detection

Aleksandar Čepić

SW, FTN Novi Sad

Definition of the problem

The purpose of this project is to count the number of times balls collide with the left and right wall in the video game "Brick Breaker" using the Jupyter Notebook platform. For the purpose of this project we were provided prerecorded videos of the game and the correct solution of each video in a separate file.




```
file,count
video1.mp4,7
video2.mp4,18
video3.mp4,21
video4.mp4,18
video5.mp4,10
video6.mp4,32
video7.mp4,13
video8.mp4,15
video9.mp4,14
video10.mp4,24
```

Image 1 and 2: Videos along with their corresponding solutions

Methodology overview

Step 1 – Working with videos

In order to detect collision between balls and the walls we first must load the video into our project after which we can access each frame of the video and extract the information we need.

We achieve this using the VideoCapture() method from the cv2 library and passing it the path to the video after which we can access the video frame by frame using the read() method.

Step 2 – Wall detection

Before we detect the position of the walls we must first preprocess the image in order to extract information about all of the edges in the image. For this purpose we used the Canny() method from the cv2 library. This method transforms the black and white image passed to it to an image with highlighted edges.

Using this preprocessed image we can now use the HoughLinesP() method to detect all the strait lines in our image. This method uses Hough transformation which keeps track of all the intersection between curves of every point in the image. If the number of intersections is above some *threshold*, then it declares it as a line and adds it to the output array in the form of (x0, y0, x1, y1).

Now that we have a list of all the lines in the frame we simply compare them and take the leftmost and rightmost lines. If the process was successful we should be able to plot the results and get an image like this.

Image 3: Blue lines highlighting the left and right wall

Step 3 – Ball detection

The first step in ball detection is to convert our frame to a binary representation where every pixel is either completely black or completely white. To achieve this we use the treshold() method with a value of 80 for the treshold. This means that every pixel with a value above 80 will be black and the others will be white.

With our image preprocessed we can now call the findContours() method to extract all the contours in the image and return them as a Numpy array of (x, y) coordinates of boundary points representing each contour.

After this we use the method minEnclosingCircle() which gives us a (x, y) pair representing the center and a radius of the circle which covers the contour with the minimum area. To check if these coordinates are in fact the ball we are looking for we simply ask weather the radius is in range of 3.6 and 5 since the balls are somewhere around this size.

Step 4 – Collision detection

Having access to the coordinates of both walls and the balls we simply check weather the distance between the center of a ball and the wall is lower or equal to a certain value and increment the counter

Result analysis

To determinate weather our analysis was successful we used the mean absolute error metric to compare our results with the correct results which were provided to us. Since our MAE is equal to 0.1 we can conclude that our implementation was succesful.