BLG 453E

Term Project Report

Team EGE

Yavuz Ege Okumuş - 150160118
Egehan Orta - 150160124

Prerequisites

In this project a monitor with a resolution of ‘1920 x 1080’ pixel has been used.

Python version 3.7.7 has been used and the following libraries and model are utilized and should be downloaded in order to run the project:

- numpy
- cv2
- nibabel
- matplotlib
- itertools
- moviepy
- scipy
- pyautogui
- dlib
- os
- time
- shape_predictor_68_face_landmarks.dat

Download link for the shape_predictor_68_face_landmarks.dat model:

http://dlib.net/files/shape_predictor_68_face_landmarks.dat.bz2

Dice Game

This game consists of three lanes, in which a dice is thrown at each, which can be seen in Figure 1. We were supposed to make the computer determine the highest valued dice and after finding it the computer must post it's result by pressing the appropriate key. In order to find out which dice had the highest value we used different computer vision techniques and algorithms.

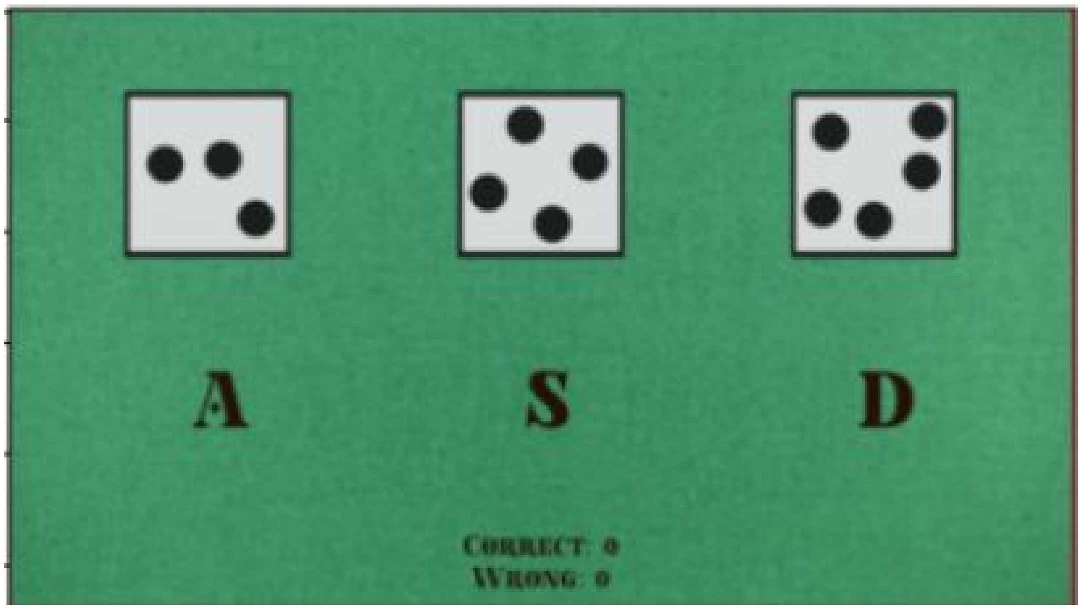


Figure 1: Dice Game Part 1

The first part of the game had the static images of dice so a screenshot of the image has been taken to be able to work on the game. In order to determine the circle numbers in it firstly we changed the image to a grayscale one to handle the issue better since in an rgb image '3 x 255' pixel values should be handled but a grayscale image has a channel so the pixels we were dealing with are reduced. This also helps us use the known algorithms with even less error since they perform better in grayscale images. In order to reduce the pixel values we handle even further a thresholding is applied to the image that eliminates the gray areas which concludes with an image made by only '0' and '255' pixel values. So that the problem is simplified, it is time to dive deep into the problem, therefore the edges have been detected using 'Canny Edge Detection'. After having an edge map of the image the contours have been found. In order to make the dots in the dice more visible, morphological operations have been made. The image we retrieved after these operations can be seen in Figure 2.

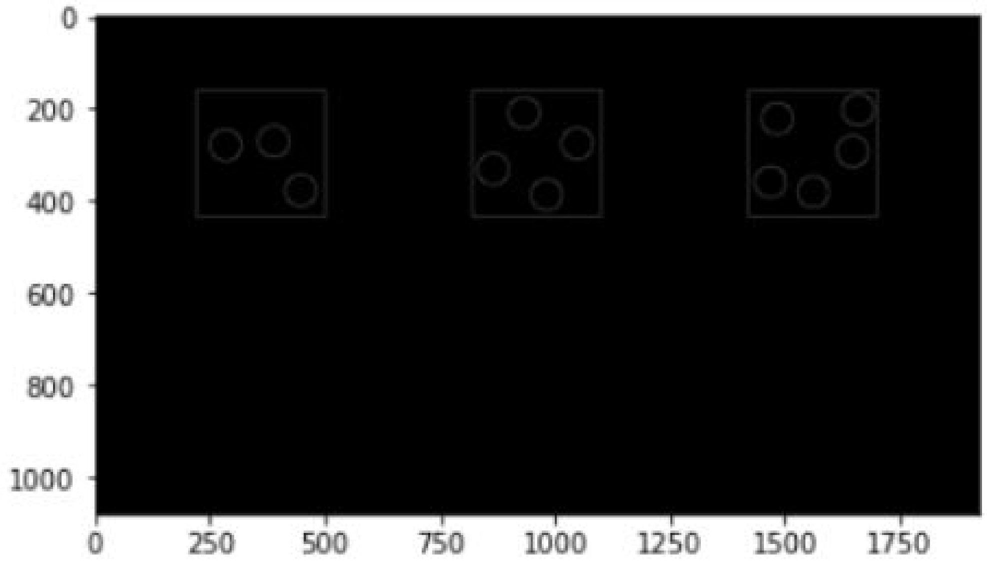


Figure 2: Detected Edges Dice Game Part 1

By having an image consisting of all the circles in a dice now it was time to count them. 'Hough Circle Detection' algorithm helped us count the circles and the counts have been compared in order to find the highest of all. Then the appropriate key is pressed and it can be said that this task is over.

The second part of the game was more challenging since the images were not static but dynamic. They perform a rotation movement which requires a different approach to the problem. This part of the game can be seen in Figure 3.

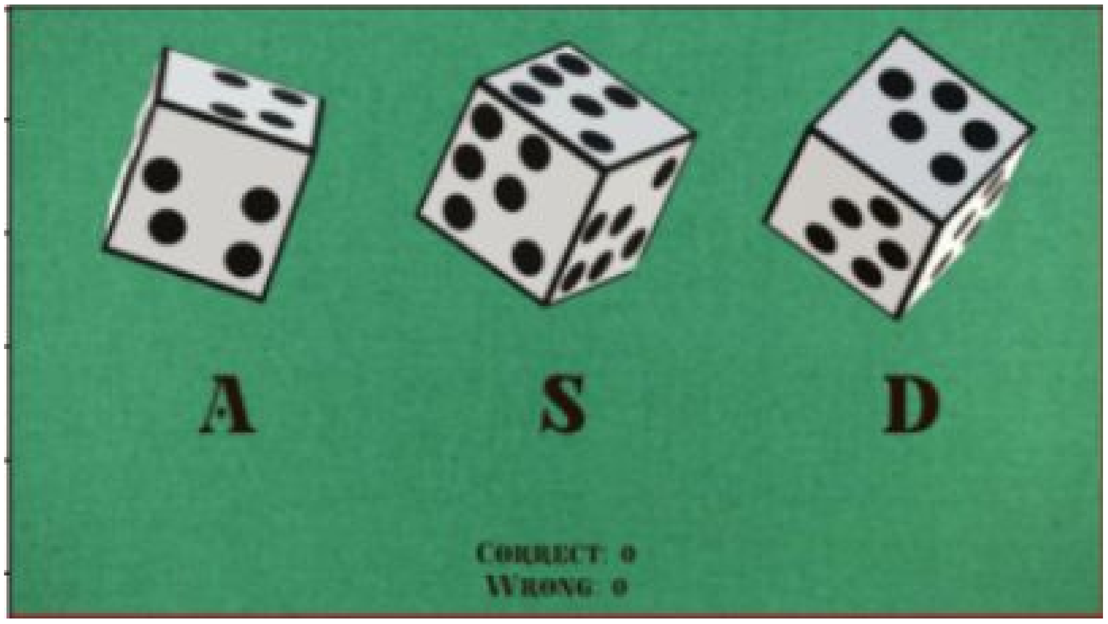


Figure 3: Dice Game Part 2

Firstly the gray scaling and thresholding operations were done again to simplify the problem. Since every dice needed better attention this time, every lane was investigated individually and since every dice was in the same position even though the circle counts were different, the screen size values to separate the game field were hard-coded. Therefore playing the game in a different resolution than what we have indicated may result in failure. After having three different images consisting of the dice, we found the contours in the images that we have seperated. An area threshold value to overlook the blurred areas, which were caused by the rotation movement, have been selected and the areas which were lower than this threshold have been ignored to find the optimum side of the dice to determine the circle count. A mask has been applied to this side to find the corner coordinates, which results in a completely white area with the size equal to the largest visible side of the dice which can be seen in Figure 4 and the version of it before these operations can be seen in Figure 5.

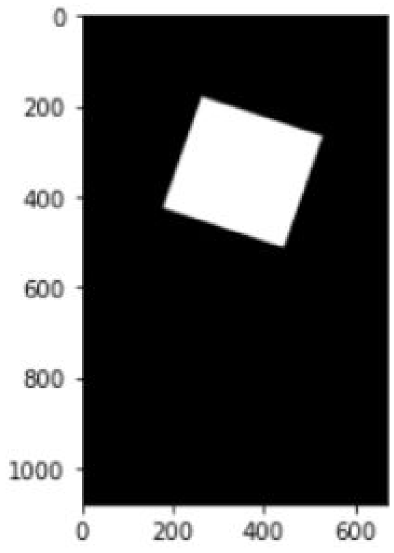


Figure 4: Masked Side

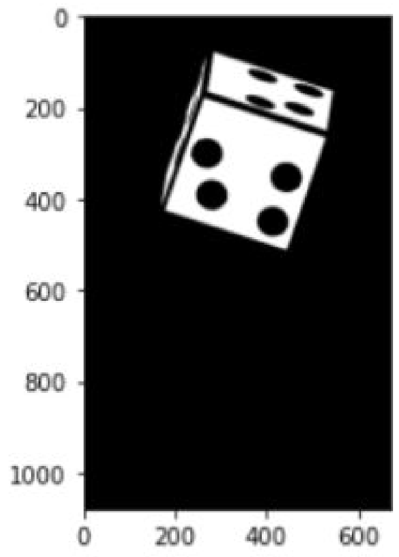


Figure 5: Before Masking

Then the ‘Shi Tomasi Corner Detection’ was applied to this image to find the corners of this newly created image. As seen in the image the surface of the dice was not a square so a transformation algorithm was used to create a ‘300 x 300’ square on the real image by using the coordinates retrieved from the corner detection step. The transformed and edge detected dice can be seen in Figure 6.

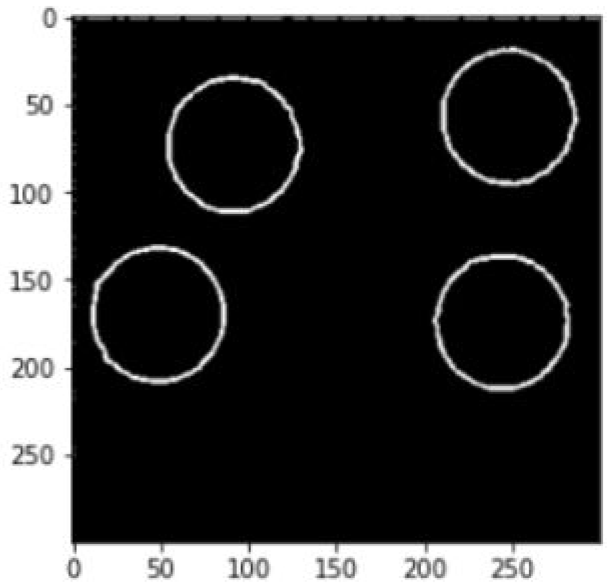


Figure 6: Transformed Dice

In conclusion we retrieved an image just like the ones we had in the first part. Therefore counting the circles on this side of the dice became the same task we handled in the first part. Simply the same algorithms have been used to detect and count them and the appropriate key presses have been simulated.

In the end our bots played the game for a time with no errors and our scores after playing can be seen in Figure 7.

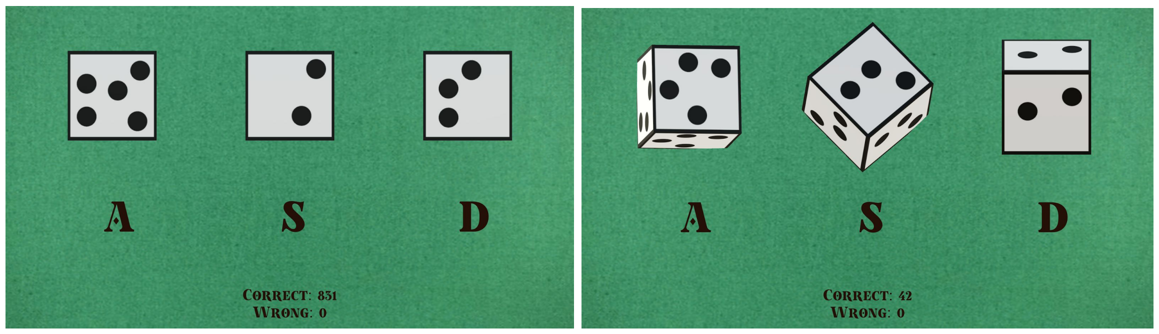


Figure 7: Gameplay Records Dice Game

Mine Game

In this game the player is on a field full of mines and our goal is to reach the final lane without touching any mines. To inform the player that he/she is headed towards a square with a mine, there is a face with some reactions on the right-bottom of the screen. Our aim is to accomplish this task autonomously with identifying whether we are heading towards a mine field or not by using various computer vision algorithms. The game can be seen in Figure 8.

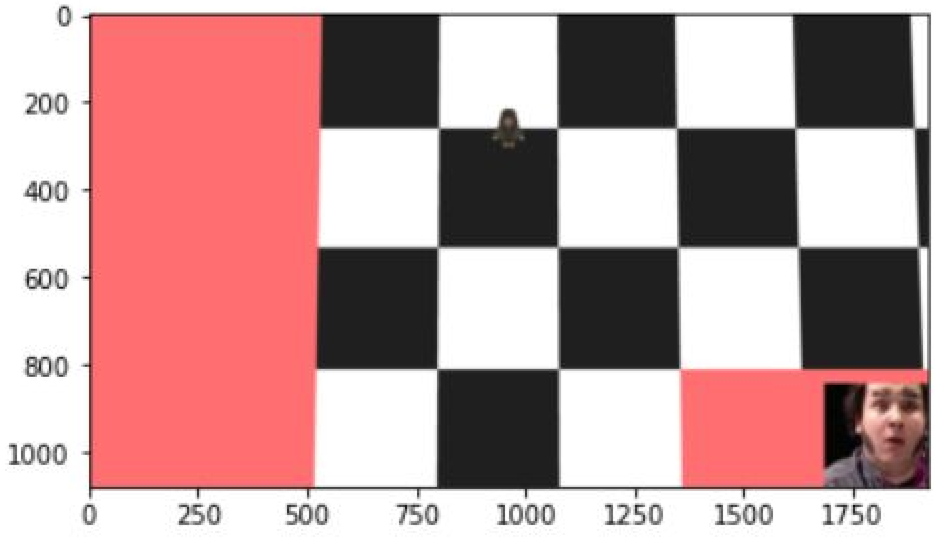
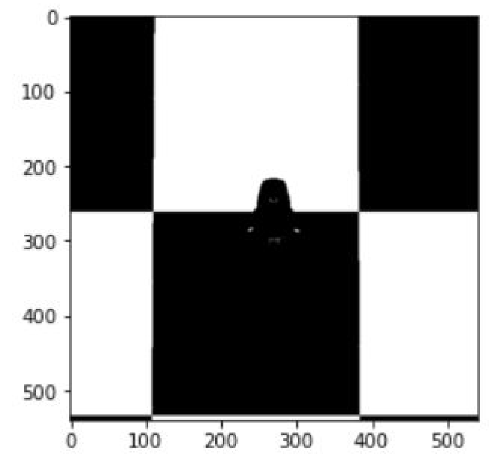


Figure 8: Mine Game Screenshot

The hardship of this task was to identify the emotions of the mine indicator guy and to not lose where our player is located. To solve these problems, we labeled the checkpoints to reach with red and safe points with green and colored them on the image. Firstly, the starting point(middle of the square) of the player is marked with red and from this location the player starts to head towards all directions which do not have any pink place after the edges of the square which the player is located on. The edges of the square are marked with green as well since they are the furthest point without stepping on a mine. Then the player starts heading towards the directions till the green areas. If the indicator guy is in shock, the player goes back to the red point and tries going in another direction till it finds a green spot where the guy is not in shock. When that point is found, it indicates that the next square is safe to pass and this square is marked with red since it is our new checkpoint and the process goes on till the player reaches the final lane.

In order to achieve these steps firstly the square that we are on should be identified. A screenshot is being taken and since the starting location of the player is known we capture the square that the player is on hard-codedly(270 x 270). A grayscale image is retrieved and pink spots in it have been found and stored. Then a thresholding is applied to turn the image into a black/white image. To be able to understand the pink spaces those pink spots we stored are recolored to pink again. So we had an image consisting of black, white and pink(grayscale). These operations are done in order to be able to label the grid even further. To distinguish each square in the grid we need to do an edge detection but since the player's icon is 3D some parts of it may conceal some edges. Therefore, some morphological operations are done to minimize the effect of the icon and the images before and after this operation can be seen in Figure 9 and 10.



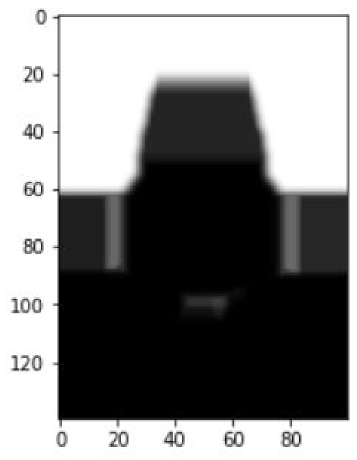


Figure 9 and 10: Mine Game before/after Morphological Operations

So that the edges are ready to be detected, 'Canny Edge Detection' is applied to the image and the result can be seen after the edge detection in Figure 11.

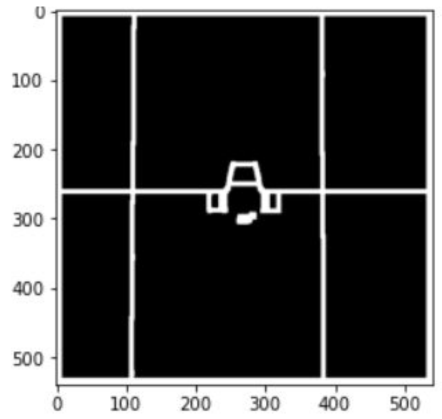


Figure 11: Mine Game after Edge Detection

So that we had the edges of the squares and the squares could be distinguished, each square is colored to a random unique color to have a clearer distinction. The colored map can be seen in Figure 12.

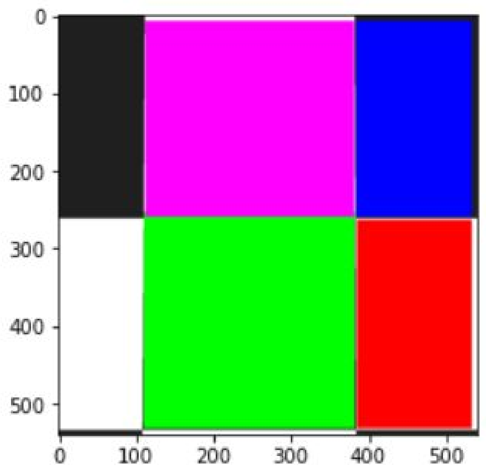


Figure 12: Mine Game after Square Distinguishment

As all squares were distinguished, now it was time to focus on the square that the player was on which is the green square in the case of Figure 12. The safe zones and the midpoint of this square were colored with appropriate colors in order to make our algorithm be applied and all other zones were painted in black since they were no longer our point of interest. This state can be seen in Figure 13.

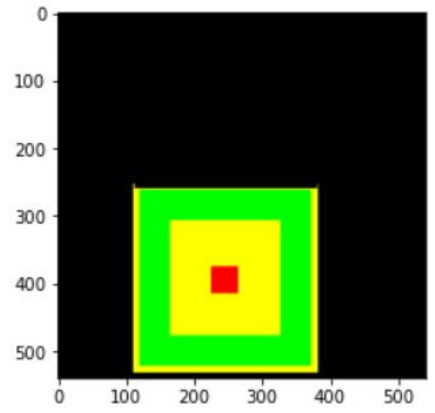


Figure 13: Mine Game after labeling the Point of Interest

Now it was time to move the player. Whenever a green spot further away from pink points is available the player tries to move on to the green zones in that direction. Upon reaching the green zones edges, the program checks the indicator guy's chin using 'dlib 68 face landmarks shape predictor' algorithm. If the chin's height is above some threshold value it indicates that there is a mine in the next square so it is not safe. So after going to the red zone back and trying different directions, if a non-shocked face was found, the player proceeds to the next square and the whole program starts working on the newest safe square. After finding the safe square in the final lane, the algorithm stops working and the game is concluded. The final state can be seen in Figure 14.

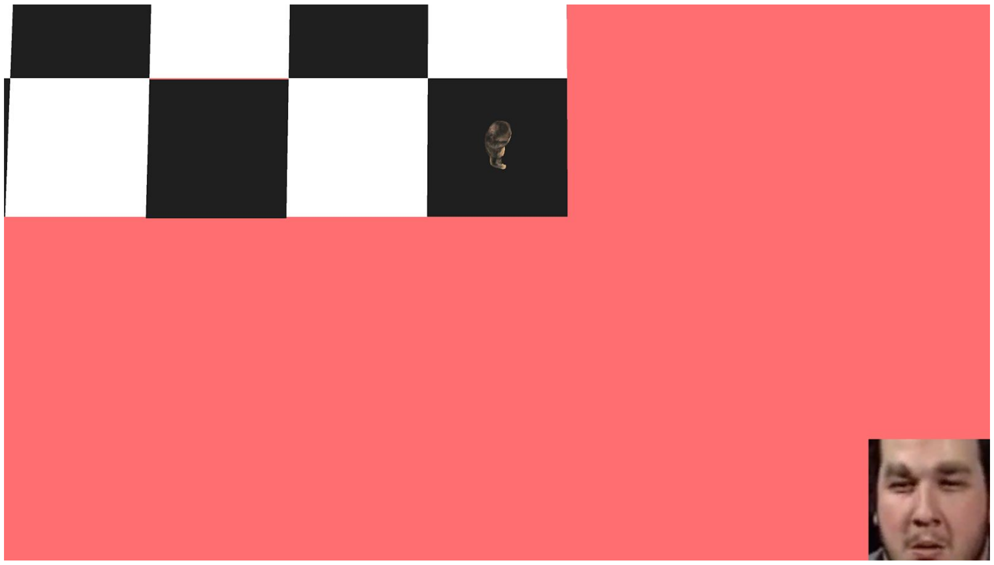


Figure 14: Mine Game - The End

Possible Bugs

1. Player moves for 0.15 seconds per iteration and the program is timeouted for a duration to make the player's velocity equal to zero while traveling to the green and red zones. If the game lags, the player may pass the zones and this may result in blasting on a mine.
2. Sometimes the indicator guy's emotions become buggy and since we are moving depending on his emotions it may mislead our algorithm. A hardcoded solution to solve this has been made. If the guy is still shocked upon reaching the middle of a square, our program understands that a bug is present and the player moves around till this emotion becomes unshocked again, but this movement can be affected by lag or different rendering times so that it may result in blasting on a mine.

Bouncing Balls

The goal of this task was to find the different colored balls and display their movement direction and calculate their average velocities in the end. We took the third homework of this class as reference to this task and used our optical flow algorithm's code snippet to achieve the goal. Overlay of the game can be seen in Figure 15.

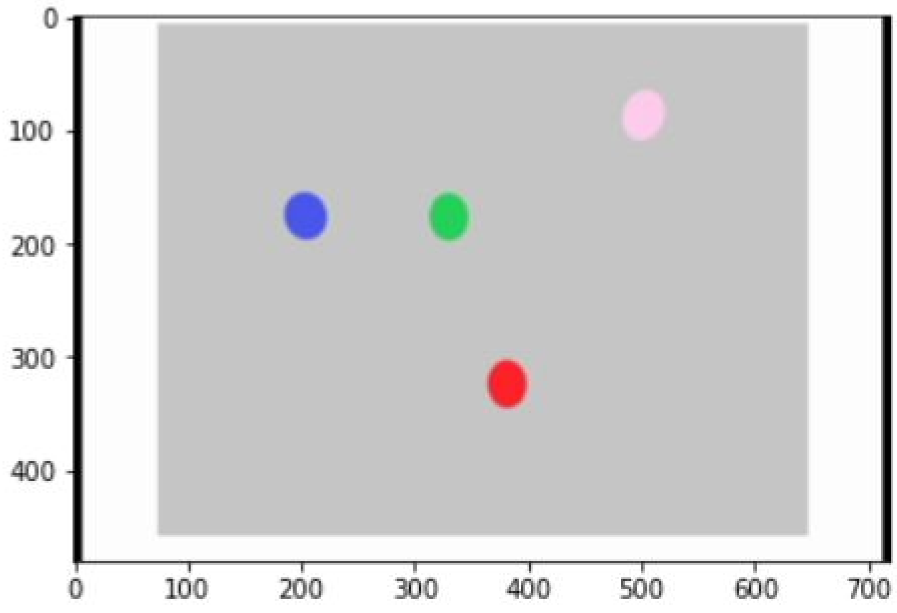


Figure 15: Bouncing Balls Game

Firstly the color space has been changed to HSV space to be able to distinguish the different colored balls more accurately. Then the boundaries of the colors have been set and a mask is applied to find the balls. The combined masked state can be seen in Figure 16.

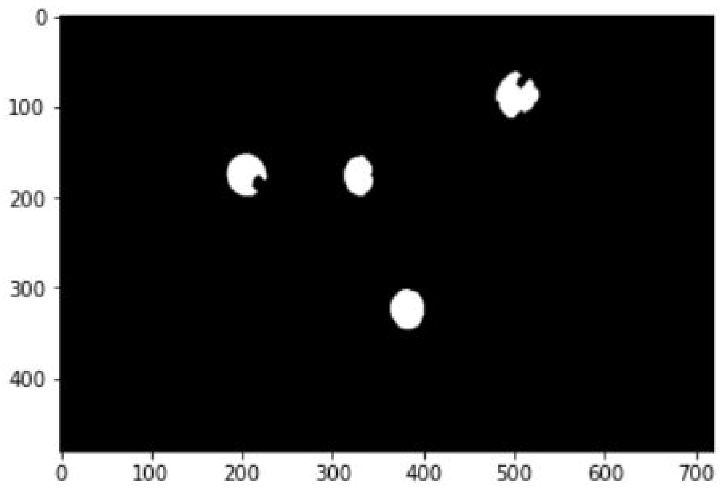


Figure 16: Bouncing Balls Masked State

After some calculations the midpoints of the balls were found and the optical flow algorithm which we implemented before is applied to the balls to find their displacement vectors. Briefly this algorithm takes frames of the game and creates a window in that frame which takes the midpoint of the balls as it’s point of interest. It compares the coordinates of the current and before midpoint and guesses which direction our ball is heading to. This results in a displacement vector. The found vectors were displayed on the screen and they can be seen in Figure 17 and the calculated average velocities in the end can be seen in Figure 18.

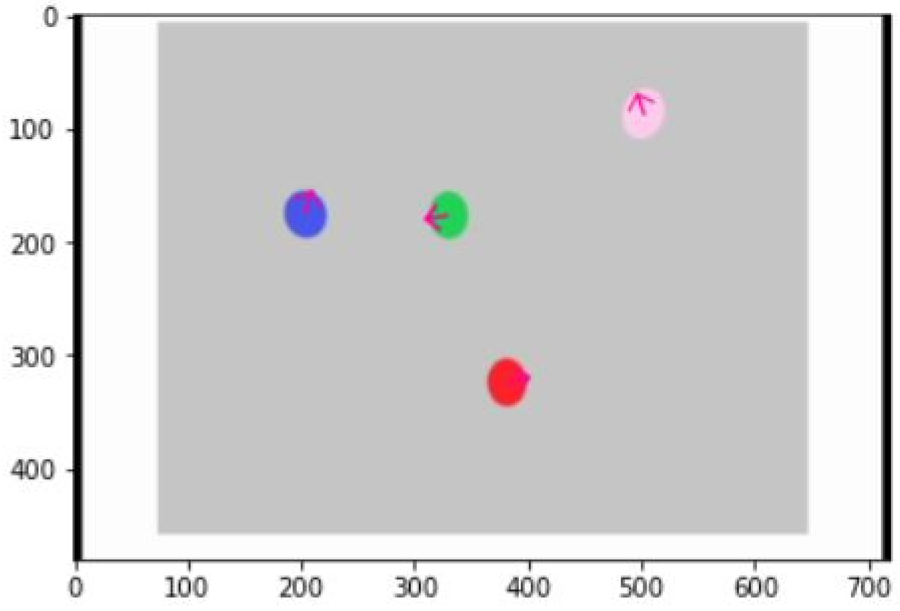
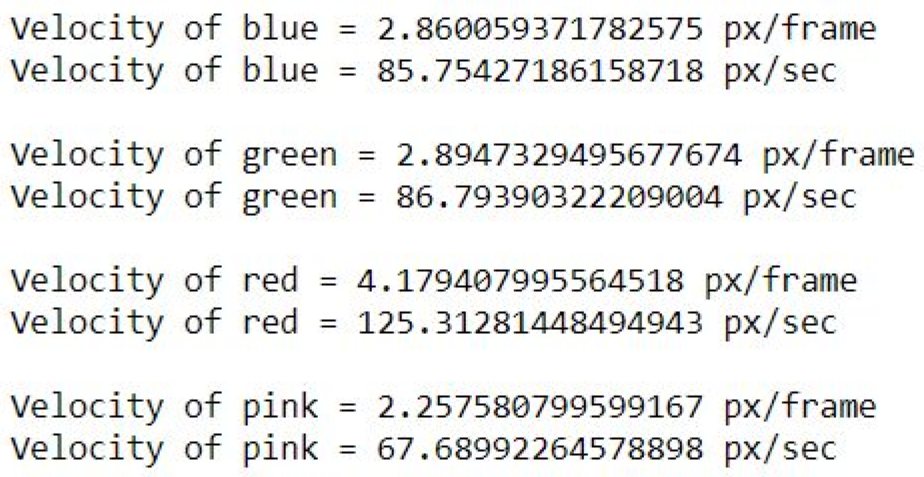


Figure 17: Bouncing Balls Displacement Vectors


```
Velocity of blue = 2.860059371782575 px/frame
Velocity of blue = 85.75427186158718 px/sec

Velocity of green = 2.8947329495677674 px/frame
Velocity of green = 86.79390322209004 px/sec

Velocity of red = 4.179407995564518 px/frame
Velocity of red = 125.31281448494943 px/sec

Velocity of pink = 2.257580799599167 px/frame
Velocity of pink = 67.68992264578898 px/sec
```

Figure 18: Bouncing Balls Ball Velocities

Vascular Segmentation

In this task we were supposed to find Poisson Noise in a 2D/3D image. To find and reduce the noise, a threshold value must have been determined to identify what is a noise and by checking the neighbors of the area we widen the investigation.

Many tests on the task have been done in order to find the optimum threshold value. After determining the threshold value a seed point has been selected and the neighbours of this area were investigated. If a point is higher than the threshold value it was labeled as a non-noisy point and the investigation expanded to that point's neighbours and added to a list. All neighbours in that list were investigated in the same way till there were no points in the list. As iterating in the list new masked images were created containing the non-noisy points in that iteration and they have been stored. When there were no points left the list all of these images brought together using bit wise or operation and the final image is retrieved. The investigations were done using 4 and 8 neighbored approaches in 2D images and 6 and 26 neighbored approaches in 3D images. The base image of the problem can be seen in Figure 19. The image we aim to retrieve can be seen in Figure 20 .The results after our approaches can be seen in Figure 21, 22, 23, 24. The accuracy of our approaches can be seen in Figure 25.

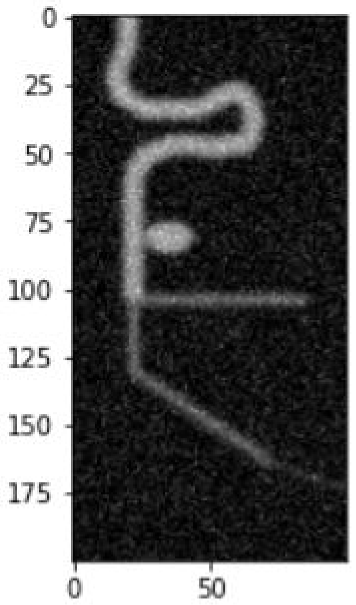


Figure 19: Vascular Segmentation Base Image

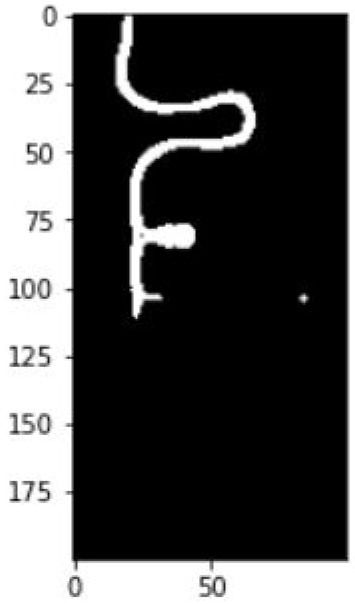


Figure 20: Vascular Segmentation Ground Truth

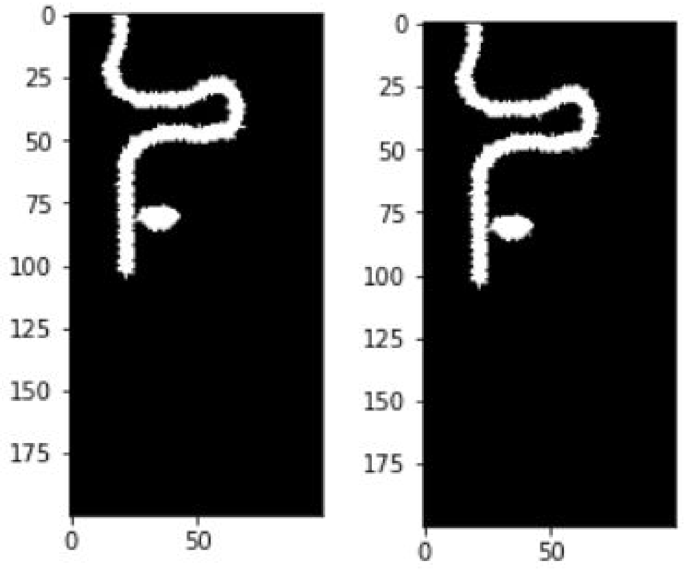


Figure 21 and 22: Vascular Segmentation 2D 4 and 8 Neighbored Approach

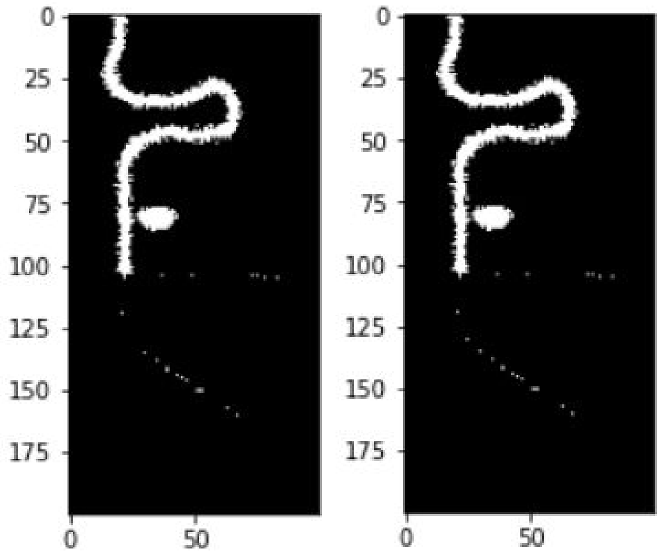
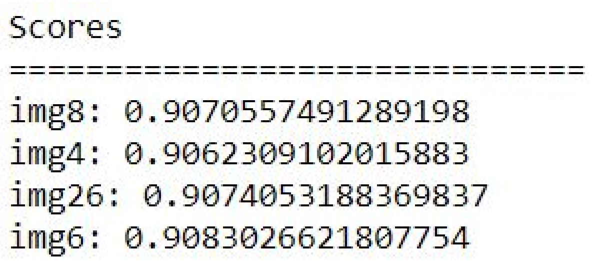


Figure 23 and 24: Vascular Segmentation 3D 6 and 26 Neighbored Approach



```
Scores
================================
img8: 0.9070557491289198
img4: 0.9062309102015883
img26: 0.9074053188369837
img6: 0.9083026621807754
```

Figure 25: Vascular Segmentation Accuracy Results