Petanque-
Game Processing And Recognition:

Final project by Yuval Rappaport & Roi Moshe

Workflow and explanations:

Petanque and our work goals

Petanque game rules:

Petanque's objective is to score the most points by having boules closer to the target than the opponent after all boules have been thrown. This is achieved by throwing or rolling boules closer to the small target ball (officially called *'Cochonnet'*) or by hitting the opponents' boules away from it. The game takes place while standing inside a circle with both feet on the ground.

Our goals:

We wanted to create a helping application for petanque players. This app will have the following capabilities for static pictures or short movies:

1. Detect all boules in the field, and divide them into teams
2. Detect the boules world position and create an up view map of the game
3. Detect the cochonnet and measure its distance from each ball.
4. Create a game status by claiming who's leading and who's turn is it

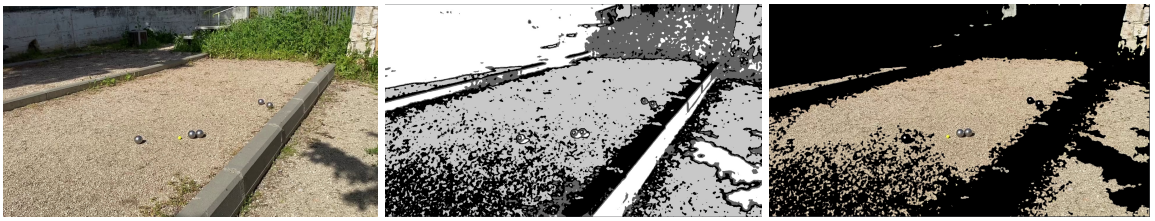
we can see a player throwing his ball as close as he can to the cochonnet

Here we can see boules from two different teams (one with grooves and one without) close to the red cochonnet

Background subtraction via K-means (segmentations detection algorithm)

In order to focus on the main playboard of the processed image and filter unnecessary noise, we wanted to mask the background. In this way we could enhance the future edge detection with more consummate parameters. The advised common way is using bilateral filtering for sharpening the edges and then K-means algorithm. While using this method we encountered numerous problems such as:

1. Problem - Devidment in the field for more than one segment due to lights/local shades/different materials of the ground-



Solution- we saw that the color of the field is unique and very different from the rest of the surroundings. Therefore we used a strong CV blur function, and divided it into less clusters (3).

Hough lines

In order to create a smaller clearer poligon for further analysis, we tried using hough lines method with the field borders to narrow the picture. In this way we could delete irrelevant parts of the masked image (the output of the first K-means stage).

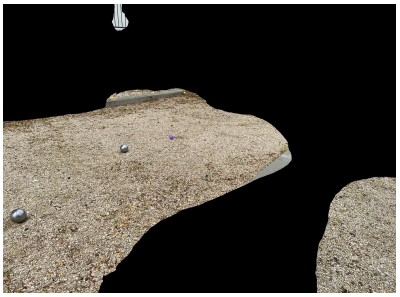
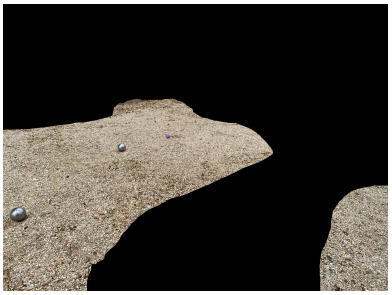
We encountered numerous problems in this part:

decision:

The results were not good enough due to implementing problems. We decided to use different methods to get only the relevant parts.

Frame burning

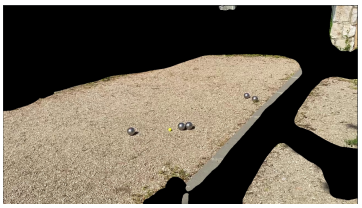
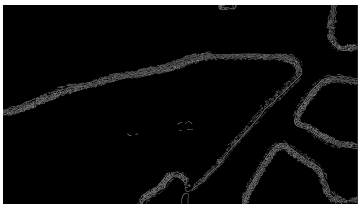
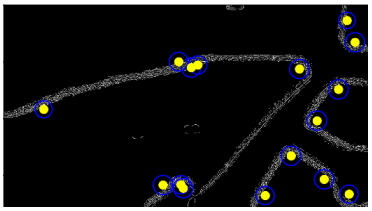
The K-means part returns the main part of the picture including the field. Because the original picture was strongly blurred we got the edge frames of the field (the concrete frames). In order to subtract them we created a burning algorithm that decreases the edges of each blob and reduces the noise of each picture.
Example:

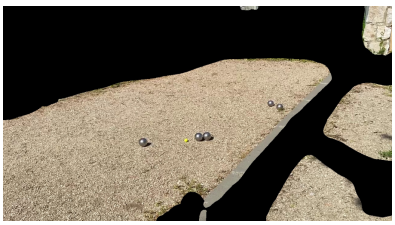
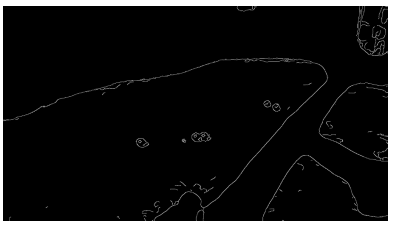
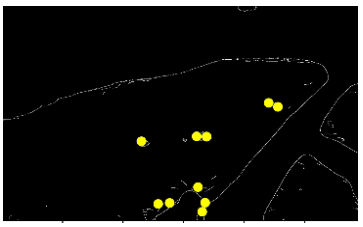
Edges detection via canny's algorithm (static pictures)

In this part we want to detect the game boules. For better results we used a masked picture of the field.
We encountered numerous problems in this part:

1. The bigger from the sides of the field were bigger than the gradients that were generated from the ball edges.

- While trying to blur the picture to soften the border edges we lost all of the edges of the picture (hence it creates a full black picture with no boules edges).
- To reduce the borders gradient we tried using different filter models (Median / Gaosian). we got better results yet still not good enough:

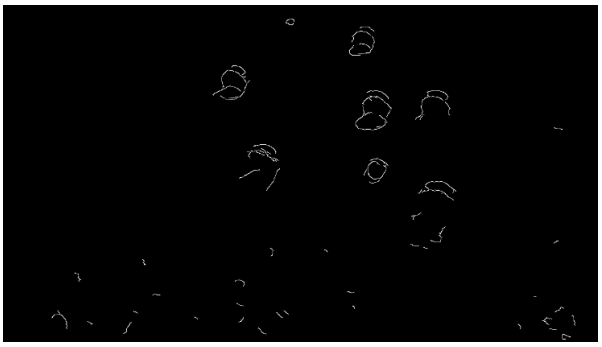




Solution: since the hough function did not differ the boules good enough from the noise in the picture we decided to implement a new hough_circle function with the few following features:

- A threshold of $0.3*2\pi*$radios for the minimum amount of pixel edges to count it as a ball.

- Prevention of multiple boules on the same area - We created an array of all the values for each pixel to be counted as a center of a ball. Later we filtered only the maximum values and compared the distances between them. We took only the largest with reasonable distance in between.
- We created a max bound of 12 boules (by the rules of the game) and took only the boules with the maximum value of edges on their circumference.

minor problem: shades or sun reflected from the boules creates edges and disrupts the outside circumference of the boules. These phenomena will always happen in certain setups.
For example:

Edges detection via PCA background subtraction

While the first edge detection was usable for static pictures, we wanted to detect edges for video as well. We used PCA background algorithm that works as follow:

1. The algorithm perceives the picture of the background before the boules are thrown, and saves it
2. boules are being thrown
3. The algorithm compares between the changes in the picture (with and without the boules) and finds the boules by the changes

examples:

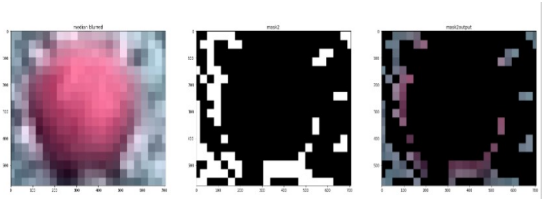
Cochonnet detection

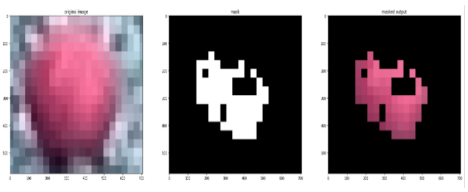
In this part we wanted to find the cochonnet. The main characteristic of the cochonnet is a relative solid color which is very different from its surroundings. Therefore we tried finding it by color. We searched on each pixel by its color and left only the pixels with color in the range of what we seek. After some tries we understood we will have to subtract ‘noise’ from other pixels than the cochonnet with the same color. Furthermore we needed to ‘strogify’ the

colors of the cochonnet. Therefore we decided to convolute the picture with a small kernel to enhance the cochonnet area and delete noise.

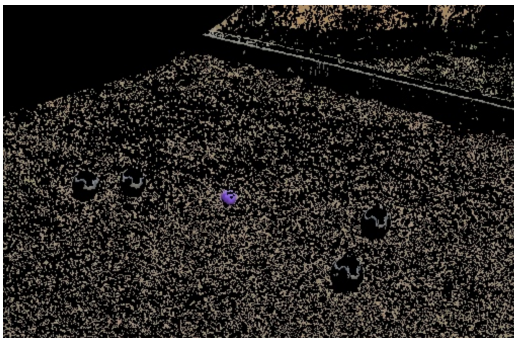
We encountered numerous problems in this part:

1. Different values for colors in files format (the following examples are the same code while the picture format is PNG and JPEG):





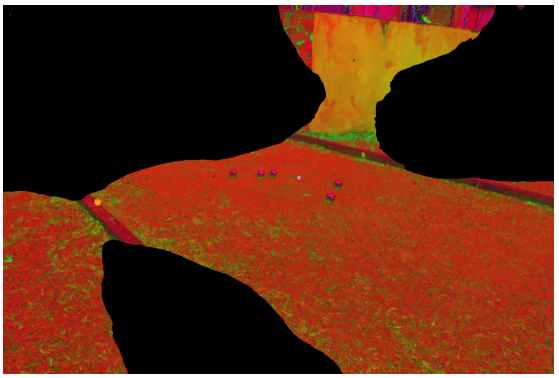
2. Finding the exact spectrum of purple:
   In this example we see what our program detected as purple (and the rest is masked in black). Our detector finds the colors of the cochonnet but also the gravel's color as purple.



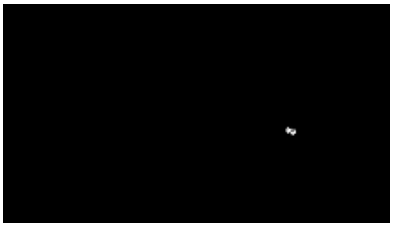
Results:

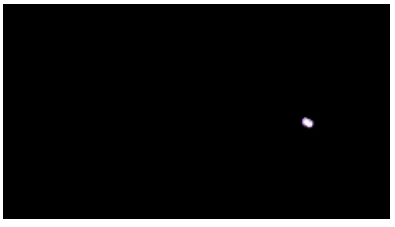
Original picture

.



presentation in HSV coordinates after K-means background subtraction





Masked picture while only the purple coordinates are shown

convoluting the last picture to enhance the cochonnet area and to cancel 'noise'

Final results:

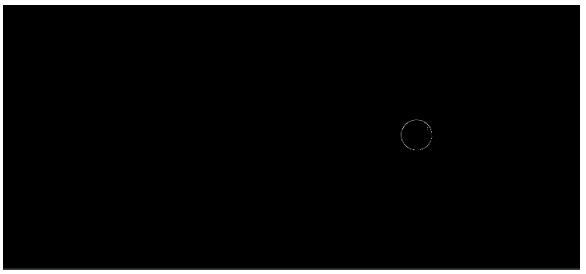
Team boules detection

After locating the boules in the picture we wanted to divide them into teams. It is possible to see that one of the teams will have grooved boules while the other smooth boules. Therefore we wanted to use edge detection to find the boules that will have more edges inside.

The main problem was to find suitable parameters for the edge detection and differentiate between edges inside boules to outside edges. In order to achieve good results we:

1. masked the entire picture for each ball
2. In order to clean noises from wrong ball detection we masked 20% of the outside layer of the found ball radius
3. We measured the amount of edges inside each ball and normalized the values relatively to the ball area by the formula - $(sum\ of\ edges\ -\ ball's\ diameter)\ /\ ball's\ area$
4. We counted sums of over 0.2 to be the grooved team, and under 0.2 to be the smooth team

For example:

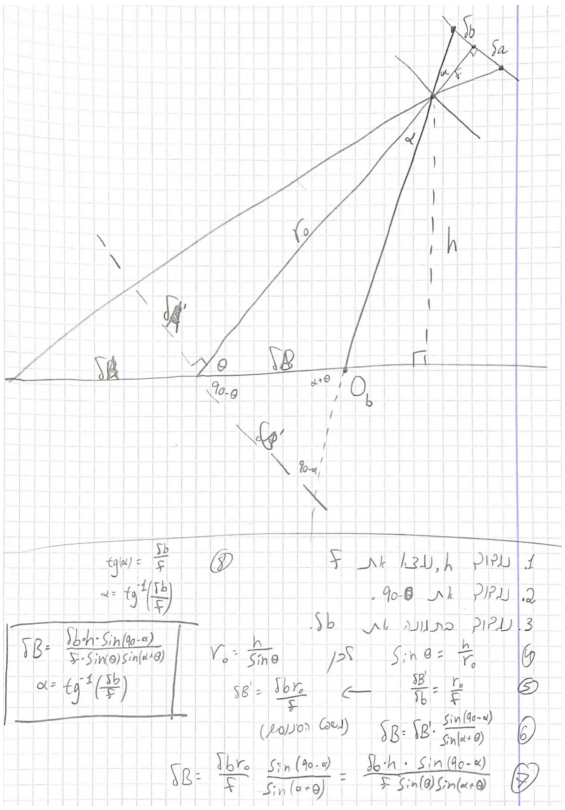


3D depth and locations geometric calculation

In this part we want to detect the game boules. For better results we used a masked picture of the field.

We encountered numerous problems in this part:

3. The bigger from the sides of the field were bigger than the gradients that were generated from the ball edges.



Final results flow

Conclusions from our work: