Complementary information on the DEADPOOL project

Requirement for the project

This project was coded with python 3.9.2 and OpenCV 4.5. Any version of python3 should be supported. However, it cannot be certified that everything will be working correctly

These libraries need to be installed / imported before compiling the project :

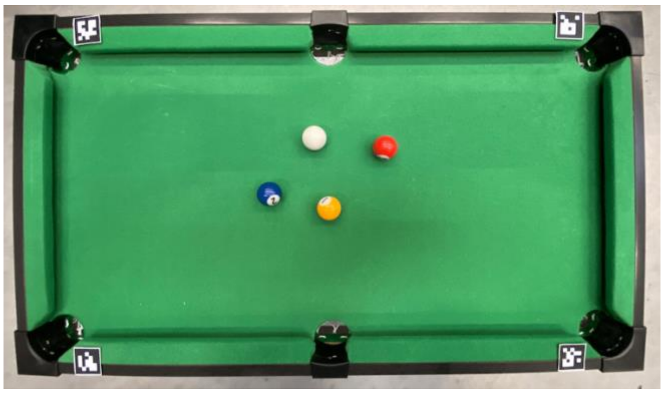
- Json
- Numpy
- Tabulate
- Sys
- Datetime
- Time
- Cv2 (OpenCv)
- Python3

How to setup the project

The projector should be set above the pool table with the projection surface covering the whole table. It is important that everything is covered. Try to center the projector and to have a correct projection. If available, try the build-in keystone correction in the projector.

The camera should be set above the pool table, in the center without obstructing the projector. Make sure the camera can see all four corners of the pool table.

The ArUCo tags must been set on the longer axis of the pool table, the closest as possible to the holes. Refer to the example below :



Once this is done and the raspberry Pi is connected to the camera, go to the next step

How to use the project

To the start the project, run the script 'startMenuDP.py' in the project folder. Everything should work directly. If not, go to the troubleshooting section.

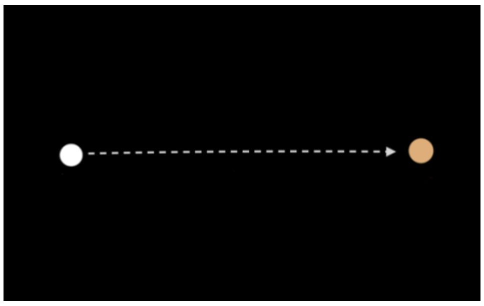
The first thing to do is to set the keystone. To do so, select the sixth option in the main menu. An image will be displayed on the pool table. Place the correct ArUCo tags on the four black and white rectangles on the image. Try to center them as much as possible. If these rectangles are not on the playable area of the pool table, recalibrate your projector.

Available games

On the images bellow, the yellow and white circles correspond to the starting zones of the balls of the same name, and the brown and cyan circles correspond respectively to the finish zone of the main ball and the secondary ball. On the figures below, the arrows represent the theoretical path of the balls and are not actually present.

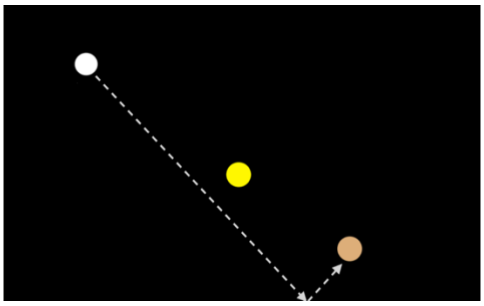
GAME 1

This game mode is the simplest mode. Only the main ball is used. There is a start zone and an end zone. The goal is to get the white ball into the finish zone as precisely as possible. The user is free to bounce or not and to specify the size of the finish zone.



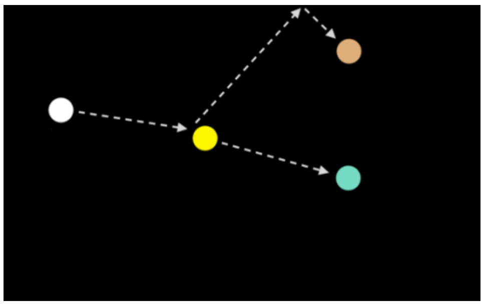
GAME 2

This looks like the first one however this time both balls are used. The main ball must always hit a target, but the second ball, the yellow ball, cannot move from its location. The current mode plans to place the secondary ball between its initial and final point to create an obstacle, but again, the possibilities are great.



GAME 3

This game mode is the most complex and perhaps the most complete. It provides for the use of two balls and two arrival zones. Currently the game is designed for the player to hit the yellow ball with the white ball so that the first one arrives in the cyan zone and the second in the brown zone.



Modify the games

To modify the games, you should make sure that you are using the same number of targets as the modified and the same color. The easiest thing to do is to use an image editing software and to move the targets. If not, here are some recommendations:

- Make sure the size of the targets is the same or larger as the old one. If the targets are too small, they might not be detected
- RGB value for the colors
    - WHITE : 255, 255, 255
    - YELLOW : 253, 247, 56
    - BROWN : 208, 173, 124
    - CYAN :154, 217, 193
- An image without any targets is available in the project folder, under assets/games/. The black rectangle represents the play area. Targets should be put inside the playing area.

Basics of the project

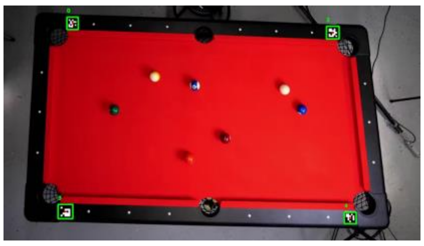
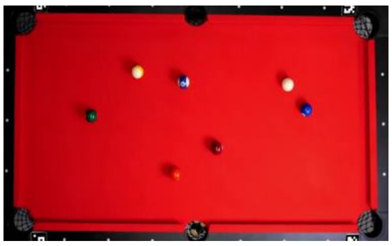
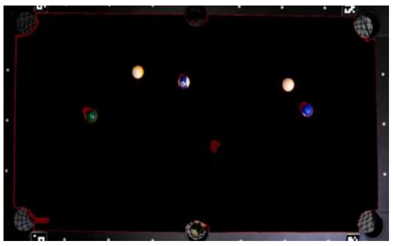
The main functions in this project are the one called “startGameX” with the ‘X’ being replaced with 1, 2 or 3. Those functions take only two input parameters, namely the path of the displayed image and the radius of the final areas.

The first step is to read this image and resize it to the overall dimensions of the project. This step is very important because right after the image is used with the perspective transformation matrix. If the images were different sizes, the transformation could not take place. Then, with the areas identified by the 'circleDetection' function, areas are placed around the final targets. The image is then saved as a “. png” file so that it can be displayed by the program immediately afterwards.

Once the image is obtained, the function displays the image in full screen on the billiard table and the user can place his marbles. As soon as the ‘Enter’ key is pressed (default key to exit an image with cv2), the program continues, and an image is taken with the camera. From there it is necessary to verify that the placement of the balls is correct. If it is, the program can continue, otherwise the previous operations are restarted.
Once this is done, the program will detect the marbles and their centers and compare them to get a score Other information is also returned for the display of the score

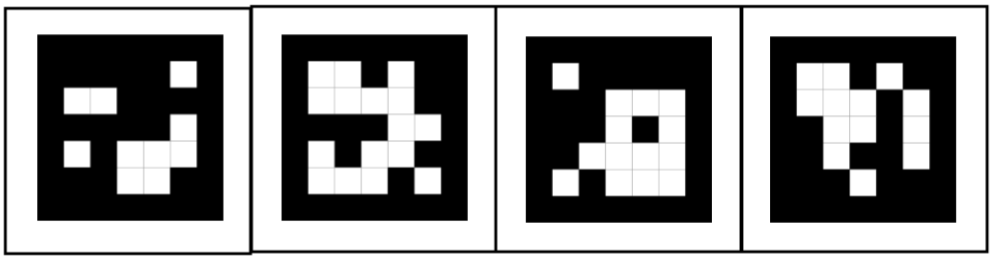
Two ‘try… except’ are used in this function to contain very specific errors. The first is to verify that the perspective correction file is present. Otherwise, it considers that it has not been done and prevents the player from playing the game. The second is there in case the detection and/or the assignment of a color to a ball is not working. This avoids a crash and don’t make the player lose his progress.

A lot of functions are hidden behind them. The main one being the circle detection for the billiard ball detection. To detect the ball, the first thing that must be done is to correct the image coming from the camera to avoid a false perspective. To do so , AruCO tags are used to detect the corners of the table and re-centered the image, onto those tags. Secondly, the background is removed for an easier detection of the balls. The detection method used is ‘cv2.HoughCircles’. But the circles are far from perfect, the ‘dp’ parameters had to be set to a very high value, thus explaining the need to remove the background. It also prevents false detection. One the billiard balls are detected another function is used to detect the main color in this circle. That is how balls are detected and assigned to a color value.

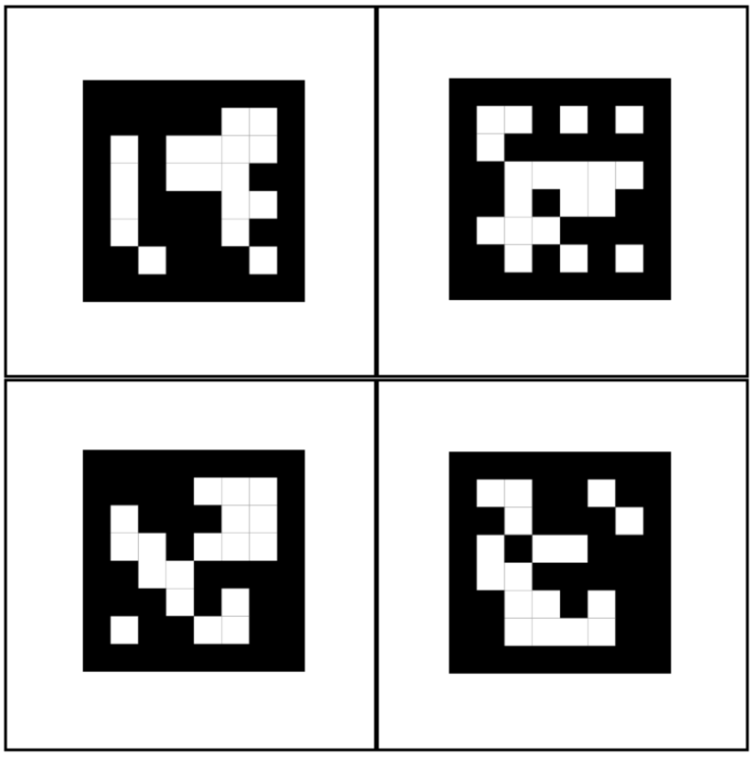




ArUCo tags

Tags used for the pool table



Tags used for the keystone



Troubleshooting

- The project is not starting
  → check the displayed error message and make sure everything is installed correctly.
  → make sure you run the correct file
  → download the file again

- The billiard ball is on the target, but the placement is not correct
  → make sure the camera rotation is correct (parameters)
  → make sure the ball is detected, an error message should be displayed. if not, add to the 'imgProcess.py' file, under the function circleDetect(), the line 'print(BGR).

- Impossible to start the game, there is an error each time
  → change the thresholds values for the color in the parameters

- Unable to the detect the camera or to take pictures
  → Make sure the ribbon cable is correctly connected on each end
  → If you've connected the camera after powering on the Raspberry, reboot the system
  →Enable the camera in the raspi-config menu

- Others…
  → contact the original creator of the project or the community if there is one