



UNIVERSITÀ DI PISA

INDUSTRIAL APPLICATION PROJECT

ACADEMIC YEAR 2021-22

License Plate Recognition

Bullari Andrea

Cancello Tortora Giuseppe

Lagna Andrea

Summary

Introduction	1
Modules/libraries	1
License plate detection	2
Character segmentation	8
Character recognition	9
Video from camera.....	10
GUI	11
Parallelization	18
Serial solution.....	18
Parallel solution	19
Serial + parallel	20

Introduction

In this documentation we will discuss about a plate recognition system that recognizes and reads License Plate Number from automobiles using Raspberry Pi and OpenCV. In particular, the program recognizes the number plate using OpenCV Contour Detection and then read the number from the plate using Tesseract OCR. Due to privacy issue, to test if the program works, we will use license from video, or picture, taken from internet. We also developed a GUI that will help the user to use the application. The application will use the plate detected to offer the user two possible games.

Modules/libraries

Here we will show some of the most important library/modules that we will need in this application:

- We will be using the *OpenCV Library* to detect the plate from the image that is given as input to the system. In particular, it is used to find all the contours in the image. It finds the bounding rectangle of every contour and then detects the plate. It's also used to open and analyze image frames or take video from the camera.
- We will use the *Tesseract OCR* library to recognize the number of the plate from the image returned by the OpenCV Library.
- We will use the *OS module*. The OS module provides a portable way of using operating system dependent functionality. In our case it will be used to create the folder where the frame from the video will be stored.
- We will need the *Picamera* module to use the Pi Camera and take picture from the raspberry.
- We will use the *Tkinter* module to create the GUI for the application.
- We will use the *imutils* library to make essential image processing functions such as translation, rotation, resizing, skeletonization, and displaying Matplotlib images easier with OpenCV.

License plate detection

The first step in this Raspberry Pi License Plate Reader is to detect the License Plate. Let's take a sample image of a car and start with detecting the License Plate on that car. The test image that I am using for this tutorial is shown below.



Step 1: Resize the image to the required size and then grayscale it. The code for the same is given below:

```
img = cv2.resize(img, (620,480) )  
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY) #convert to grey scale
```

The image would be transformed something like this when this step is done:



Step 2: Every image will have useful and useless information, in this case for us only the license plate is the useful information the rest are pretty much useless for our program. This useless information is called noise. Normally using a bilateral filter (Blurring) will remove the unwanted details from an image. The code for the same is

```
gray = cv2.bilateralFilter(gray, 11, 17, 17)
```

The output image is shown below, as you can see the background details (tree and building) are blurred in this image. This way we can avoid the program from concentrating on these regions later.



Step 3: Now we can start looking for contours on our image.

```
nts = cv2.findContours(edged.copy(), cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
cnts = imutils.grab_contours(cnts)
cnts = sorted(cnts, key = cv2.contourArea, reverse = True)[:10]
screenCnt = None
```

Once the contours have been detected we sort them from big to small and consider only the first 10 results ignoring the others. In our image the contour could be anything that has a closed surface but of all the obtained results the license plate number will also be there since it is also a closed surface.

To filter the license plate image among the obtained results, we will loop through all the results and check which has a rectangle shape contour with four sides and closed figure. Since a license plate would definitely be a rectangle four-sided figure.

```
# loop over our contours

for c in cnts:

    # approximate the contour

    peri = cv2.arcLength(c, True)

    approx = cv2.approxPolyDP(c, 0.018 * peri, True)

    # if our approximated contour has four points, then

    # we can assume that we have found our screen

    if len(approx) == 4:

        screenCnt = approx

        break
```

The value 0.018 is an experimental value. Once we have found the right contour we save it in a variable called *screenCnt* and then draw a rectangle box around it to make sure we have detected the license plate correctly.



Step 5: Now what we know is where the number plate is, the remaining information is pretty much useless for us. So, we can proceed with masking the entire picture except for the place where the number plate is. The code to do the same is shown below.

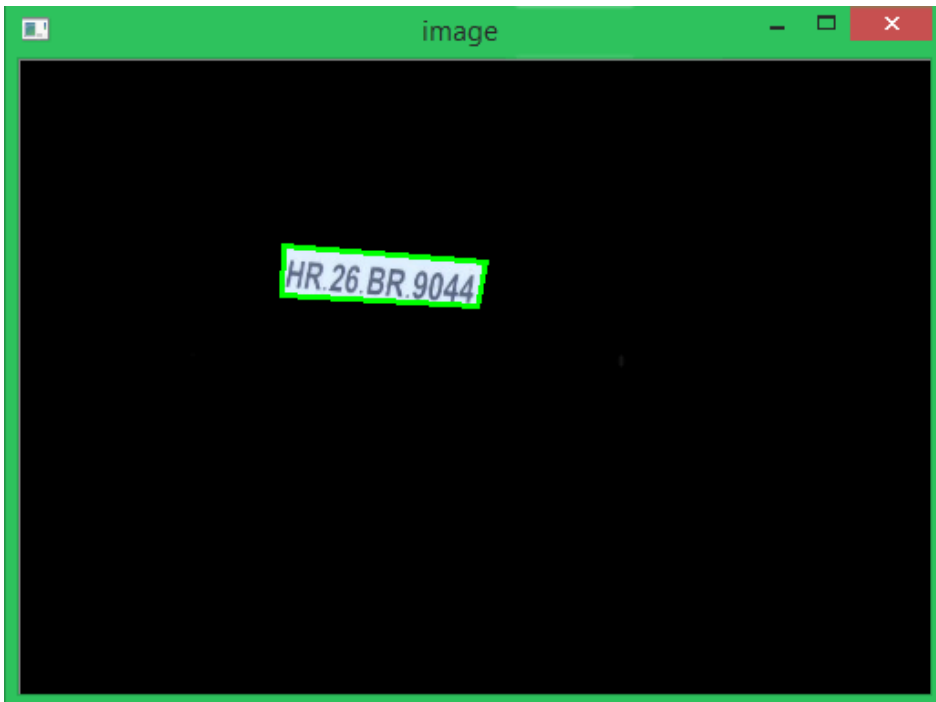
```
# Masking the part other than the number plate
```

```
mask = np.zeros(gray.shape,np.uint8)
```

```
new_image = cv2.drawContours(mask,[screenCnt],0,255,-1,)
```

```
new_image = cv2.bitwise_and(img,img,mask=mask)
```

The masked new image will appear something like below



Character segmentation

The next step in Raspberry Pi Number Plate Recognition is to segment the license plate out of the image by cropping it and saving it as a new image. We can then use this image to detect the character in it. The code to crop the Roi (Region of interest) image form the main image is shown below.

```
# Now crop  
  
(x, y) = np.where(mask == 255)  
  
(topx, topy) = (np.min(x), np.min(y))  
  
(bottomx, bottomy) = (np.max(x), np.max(y))  
  
Cropped = gray[topx:bottomx+1, topy:bottomy+1]
```

The resulting image is shown below.



Character recognition

The Final step in this Raspberry Pi Number Plate Recognition is to read the number plate information from the segmented image. We will use the *pytesseract* package to read characters from image.

We could use the `pytesseract.image_to_string` function to get directly the plate as string but we could not get the confidence levels of the OCR (Optical Character Recognition).

The `image_to_data` returns a *pandas* datagram and to get the confidence and the text we use the following code:

```
#Read the number plate
```

```
text = pytesseract.image_to_data(Cropped, output_type='data.frame', config='--psm 6')
```

```
text = text[text.conf != -1]
```

Video from camera

We can use the picamera from the raspberry to get a video from the application and then, from the video, we analyze each frame and get the frame that better recognize the plate. This is option is suggested only if you possess a high-quality camera (you need a good quality picture to get the plate from the image). To get the video from the camera we use the following code:

```
camera = "" #PiCamera()

camera.resolution = (640, 480)

camera.framerate = 30

rawCapture = ""#PiRGBArray(camera, size=(640, 480))

for frame in camera.capture_continuous(rawCapture, format="bgr",
use_video_port=True):

    image = frame.array

    cv2.imshow("Frame", image)

    key = cv2.waitKey(1) & 0xFF

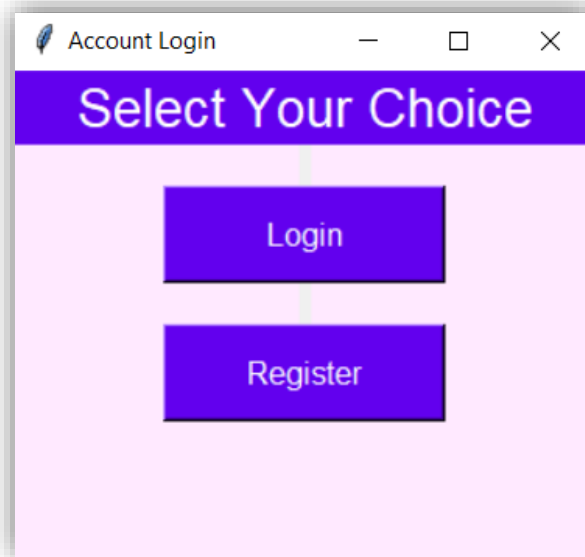
    rawCapture.truncate(0)

    if key == ord("s"):
```

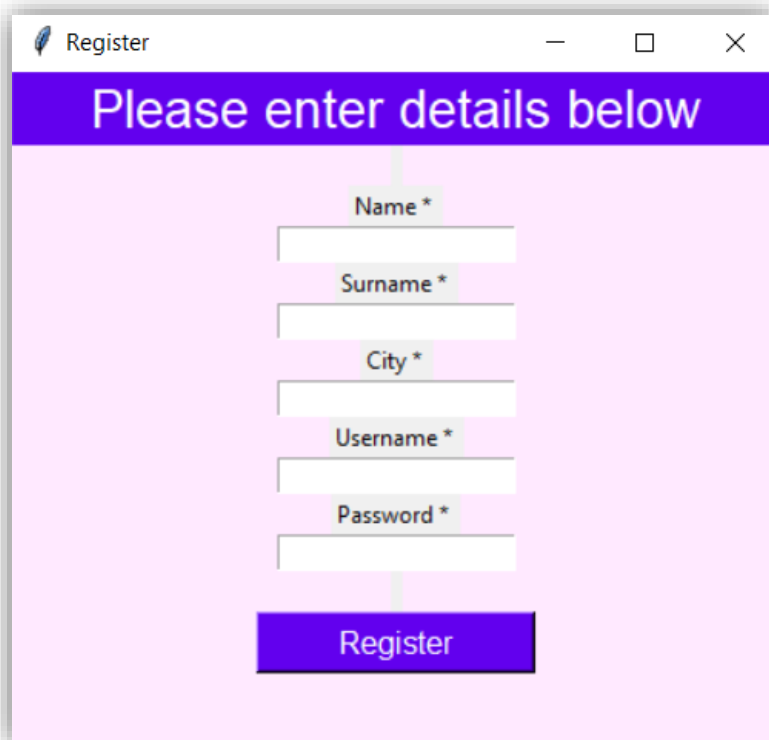
We first initialize the camera object and set the resolution at (640, 480) and the frame rate. Then use the *capture_continuous* function to start capturing the frames from the Raspberry Pi camera. We are using the keyboard key 'S' to end the video. When the keyboard key is pressed, it will take all the frames and save them to a repository called "capture" (in case the repository doesn't exists, it will be created) and then process them to select the best one (depending on the confidence level).

GUI

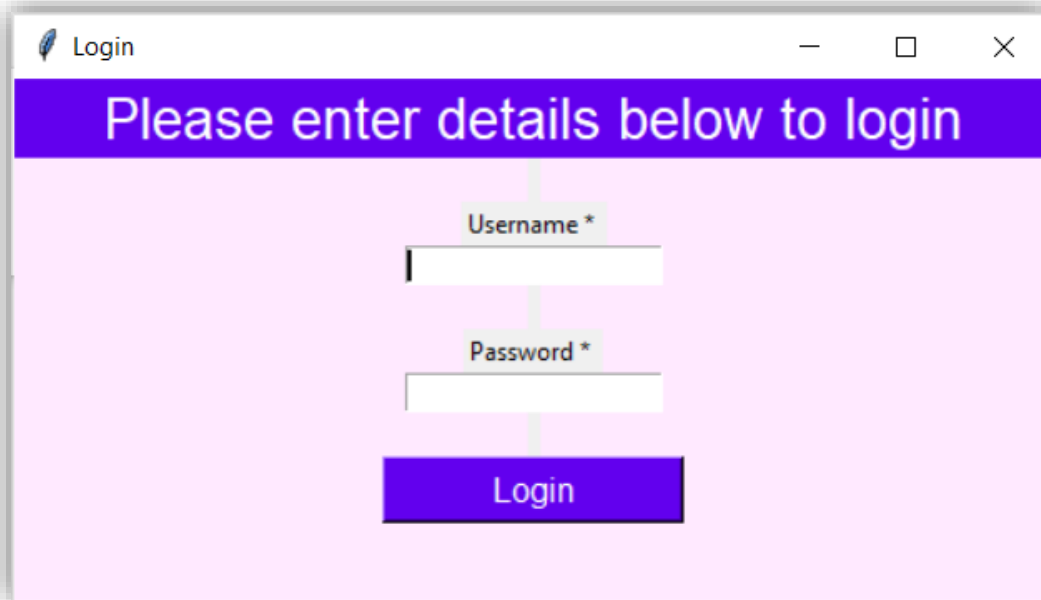
We also developed a graphic user interface to simplify the use of the application to the user. At first, the following window is shown:



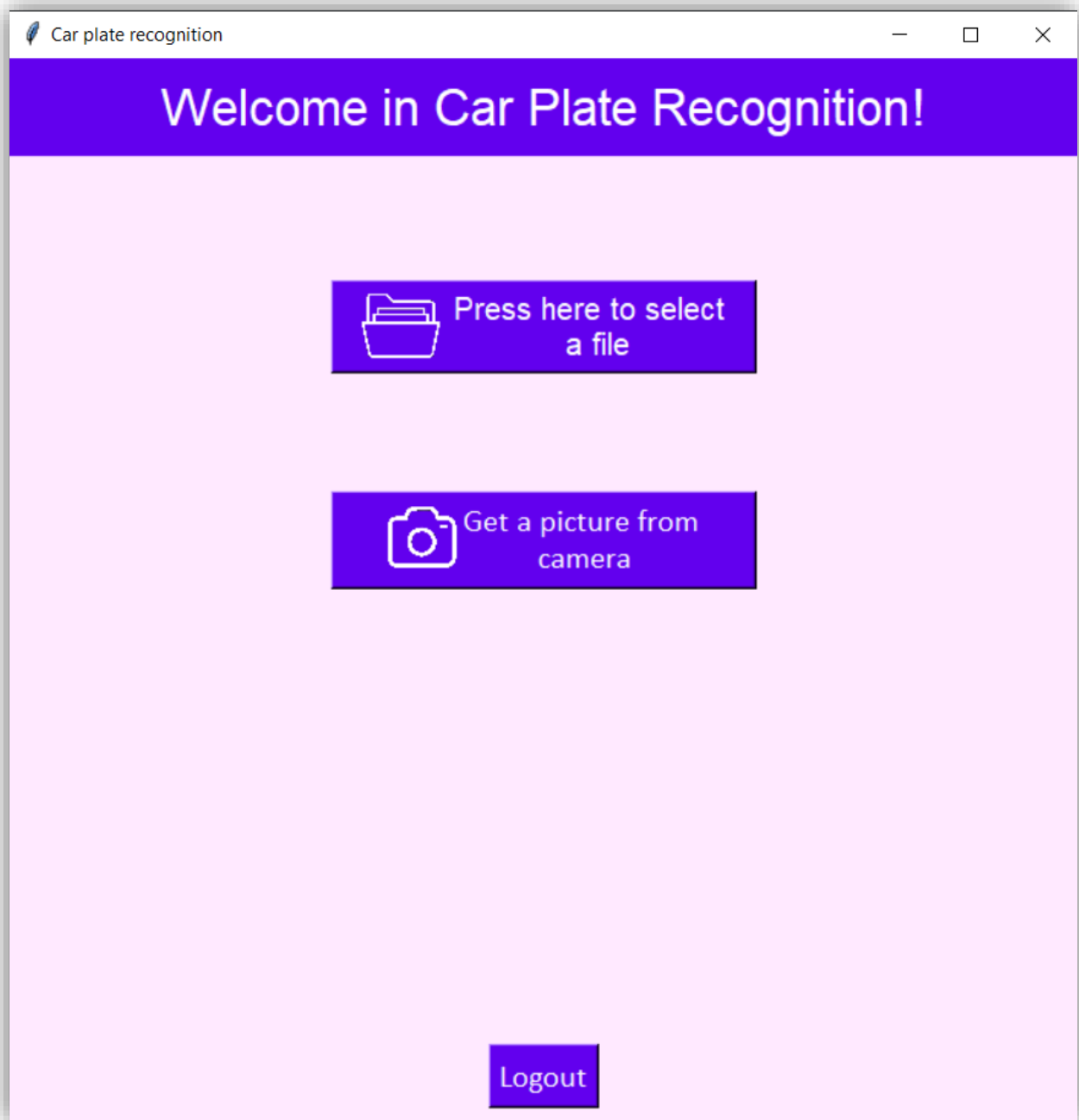
At first the system asks the user to login. If the user does not have an account, he can register (registration is mandatory to use the application). To register, the user must enter information such as name, surname, city, username and password as shown in the picture below:

A screenshot of a graphical user interface window titled "Register". The window has a standard title bar with a feather icon, a minus sign, a maximize button, and a close button. Below the title bar is a purple header bar with the text "Please enter details below" in white. The main area of the window has a light pink background. In the center, there is a registration form with six input fields, each with a label and an asterisk indicating it is required: "Name *", "Surname *", "City *", "Username *", and "Password *". The input fields are white with a light gray border. Below the input fields is a blue rectangular button with white text: "Register". A thin vertical line separates the input fields from the button.

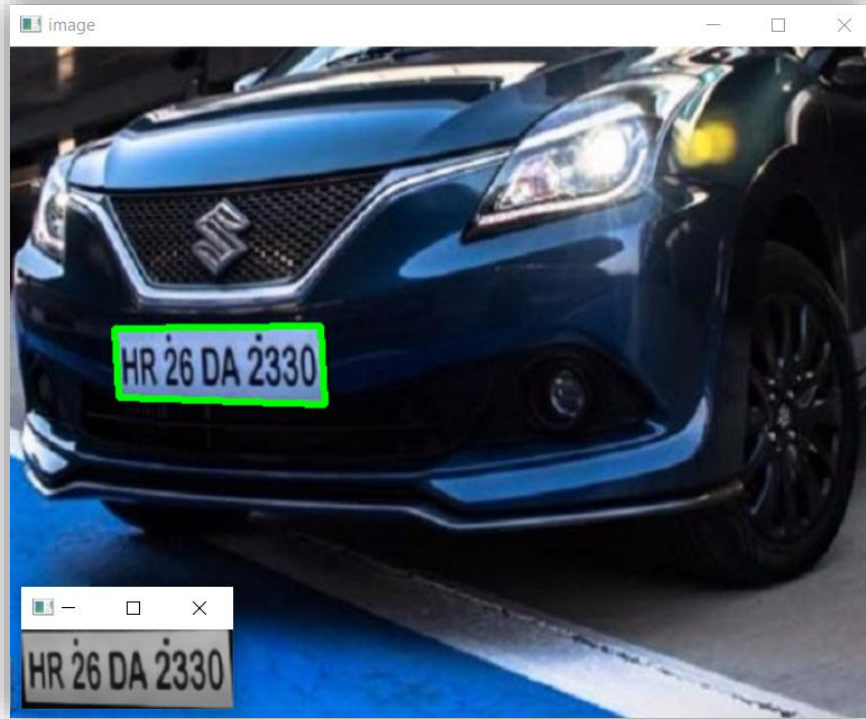
All the fields are mandatory to register. Only characters are allowed in the various fields, except for the username (where you can also enter numbers). Even if special characters are entered, these will be detected and removed from the system. After the registration, the user can login. The system will ask to enter the username and the password as shown below:

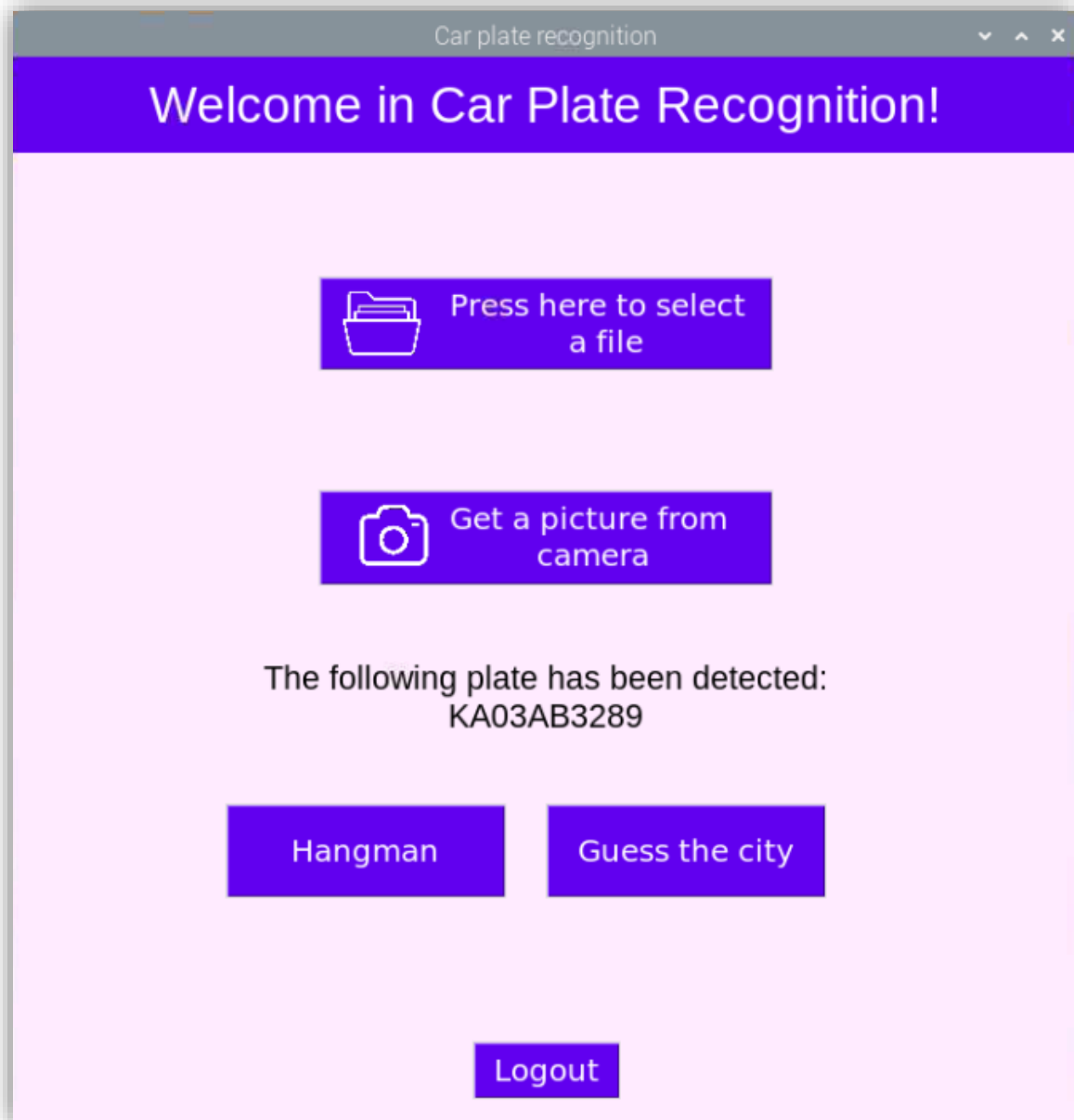


After logging in, the user can decide to start playing. To start the game, the application needs an image to get the plate number. The user can either select an image/video or can take a picture using the camera from the raspberry and detect the plate from the image.



In case the user decides to select a video, the application will analyze the video and get the frame in the video in which the plate is better recognized. To do so, we first extract from the video several frames depending on the duration of the video and the rate at which we want to capture frames. Then we analyze each frame and get the one that has the highest confidence level (the confidence level is returned by the *tesseract.image_to_data* function).





After getting the plate number, we remove all the numbers from the plate and leave only the characters, that are what we need for the games. The user can choose which game to play: "Guess the city" or "Hangman". The 'Guess the city' game consists in asking the user to digit an Italian city that contains one or more, characters recognized from the plate and then assigns a score depending on the number of characters that the city name has in common with the plate letters.

Guess the city

Please insert an Italian city that contains most of the following letters:

KAAB

Check

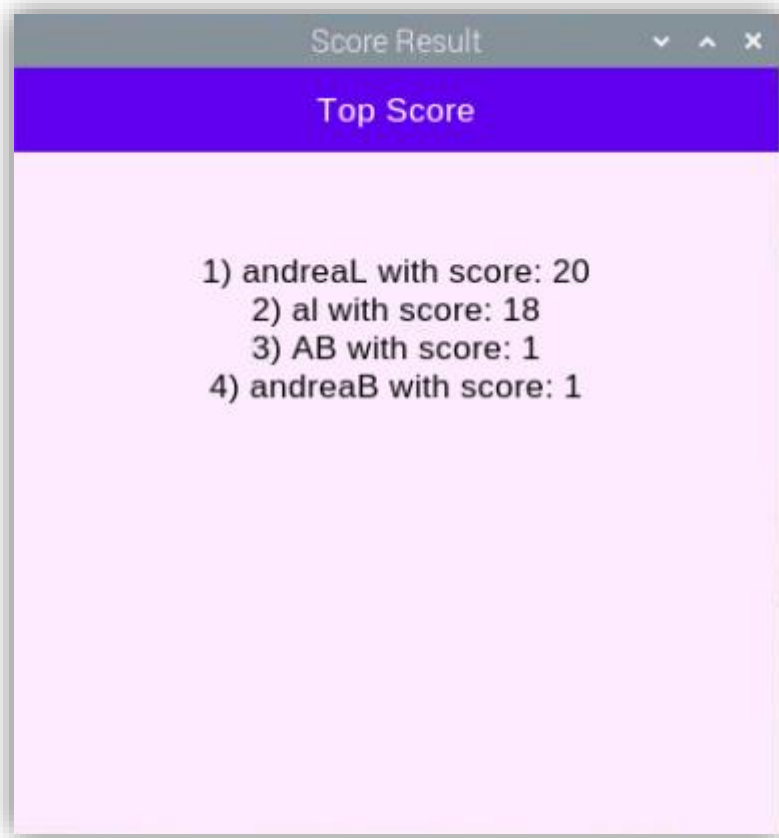
Score Result

Your score is: 4
Do you want to check the classification?

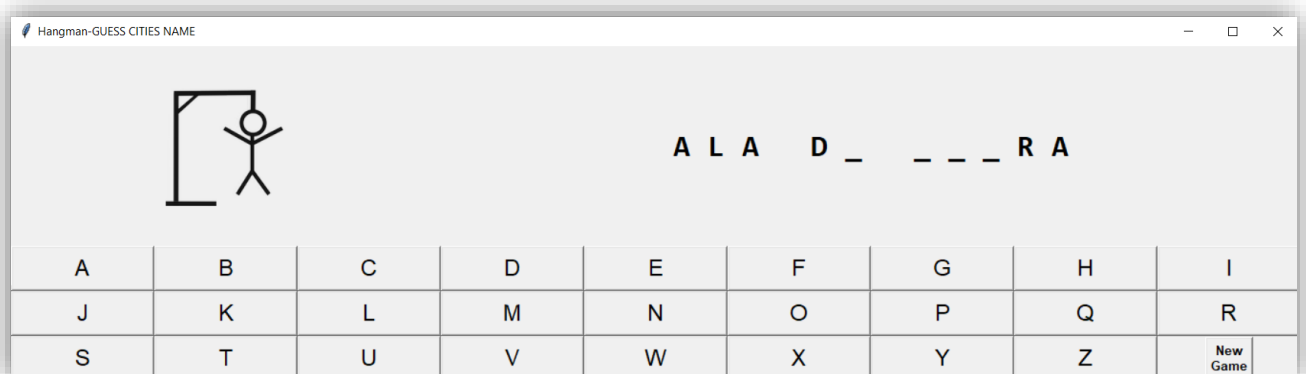
Yes

No

To keep the competition high, a ranking is also drawn up with the users who have the most points, like is shown in the picture below:



The "Hangman" game is a game in which the user must find the city randomly selected - between those that contain one or more characters in common with the detected plate - the by the system by guessing one letter at a time. If the letter is in the city, the letter appears on all matching dashes. If the letter is not present, the player loses one life, and the game draws a piece of the hangman diagram. The game ends when the word is guessed, or the hangman diagram is completed. An example of a game is shown in the picture below:



Parallelization

To improve the performance of the system, we have decided to use a parallelized approach. Obviously, the approaches that have been considered are only useful in the case of analyzing a video as the analysis times of the photo cannot be parallelized. Two different approaches have been tried to verify which of them is the best.

Serial solution

```
279 def serial_solution(camera, video_path, rate):
280     ts = time.time()
281
282     results = []
283     lastFrame = 0
284     if(camera):
285         lastFrame = get_frames_from_camera()
286     else:
287         lastFrame = frame_from_video(video_path, rate)
288
289     results = []
290     for i in range(lastFrame):
291         if camera:
292             name = "capture/frame" + str(i) + ".jpg"
293             print("prova2")
294         else:
295             name = "data/frame" + str(i) + ".jpg" ## remove .0
296         text, conf = detect_plate(name,0)
297         if(conf > 0):
298             text = ''.join([c for c in str(text) if c.isupper() or c.isdigit()])
299             if (not str(text).isnumeric() and len(str(text)) > 6 and len(str(text)) < 9):
300                 results.append({'plate':text, 'conf': conf, 'picture':i})
301
302     results.sort(key=lambda x: x.get('conf'), reverse=True)
303     best = results.pop(0)
304     picture = 0
305     print('Time in serial:', time.time() - ts)
306     if (camera):
307         detect_plate("capture/frame" + str(best.get('picture')) + ".jpg",1)
308     else:
309         detect_plate("data/frame" + str(best.get('picture')) + ".jpg",1)
310     return best.get('plate'), best.get('conf')
```

We first create an empty array “results” in which we are going to insert the plate information obtained by all the frames of the video. We check with the use of the Boolean variable “camera” if the video is obtained by the camera (using the Picamera) or by a video selected by the user. In both cases an integer value is returned and stored in the “LastFrame” variable (So we can know the number of frames). Then, we check all the frames. Then, all the frames are checked to extract the license plate (which is associated with a confidence level). Only consistent license plates are added to the results array (therefore, they need to have at least one number or one character and those that do not have a number of digits between 6 and 9 are excluded). After the extraction of the plate from the frames, we sort the result and get the plate with the highest level of confidence from the array and return the results.

Parallel solution

```
313 def parallel_solution(camera, video_path, rate):
314     ts = time.time()
315
316     lastFrame = 0
317     result = []
318     if camera:
319         lastFrame = frame_from_webcam()
320     else:
321         lastFrame = frame_from_video(video_path, rate)
322     pool = mp.Pool(mp.cpu_count())
323     result.append(pool.map(fun, range(0, lastFrame)))
324     pool.close()
325     pool.join()
326     result[0].sort(key=lambda x: x.get('conf'), reverse=True)
327     best = result[0].pop(0)
328     print('Time in parallel:', time.time() - ts)
329     detect_plate("data/frame" + str(best.get('picture')) + ".jpg", 1)
330     return best.get('plate'), best.get('conf')
331
332 def fun(i):
333     if camera:
334         name = "capture/frame" + str(i) + ".jpg"
335     else:
336         name = "data/frame" + str(i) + ".jpg"
337     text, conf = detect_plate(name, 0)
338     if (conf > 0):
339         text = ''.join([c for c in str(text) if c.isupper() or c.isdigit()])
340         if (not str(text).isnumeric() and len(str(text)) > 6 and len(str(text)) < 9):
341             return {'plate': text, 'conf': conf, 'picture': i}
342     return {'plate': "", 'conf': 0, 'picture': 0}
```

The "parallel approach", consists in parallelizing the analysis of frames of the video. In this way we can exploit all the cores of the system to reduce the time needed to extract the plate number from the frames and then take, from all the frames, the one with the highest confidence level. We exploit the [multiprocessing](#) package, that supports spawning processes, to create a pool of processes. The pool functions takes as argument the number of process to create, in this case we create as many process as the number of core in the system (we use the `mp.cpu_count()` function to get the number of cores). Then we parallelize the analysis of the frames using the `poo.map` function. At the end we close the pool and use the `join` function to wait for all the process (We need the analysis of all the frames to select the best). At the end the license plate is returned.

Serial + parallel

```
109 pool = mp.Pool(mp.cpu_count())
110 while (True):
111     # reading from frame
112     ret, frame = cam.read()
113
114
115     if ret:
116         # if video is still left continue creating images
117         name = './data/frame' + str(int(currentframe/rate)) + '.jpg'
118         # writing the extracted images
119         if(currentframe%rate == 0):
120             res = pool.apply_async(call_func, (name, frame,int(currentframe/rate)))
121             arrayFrames.append(res.get())
122             #print('Creating...' + name)
123             # increasing counter so that it will
124             # show how many frames are created
125             currentframe += 1
126         else:
127             break
128     pool.close()
129     pool.join()
```

The second approach is called “Serial + Parallel approach”. In this case, a pool of processes is created before the extraction of the frames from the video. the parallelization is performed immediately after the extraction of each frame (while in the “parallel solution”, we wait the extraction of all the frames before doing the parallelization). In the “call_func”, the processes extract the plate and the confidence level and they stored in the “arrayFrames” array (which is a global variable) used later to extract the results of the analysis and the select the plate with the highest level of confidence.

Both the “Parallel approach” and the “Serial + Parallel approach” were compared with each other and with the Serial solution (which is the one without any parallelization) and the following results were obtained:

VIDEO	DURATION	SERIAL APPROACH	PARALLEL APPROACH	SERIAL + PARALLEL APPROACH
VideoAudi.mp4	0:11	6 sec	5 sec	8 sec
Video.MOV	0:26	22 sec	15 sec	24 sec
VideoCar	1:48	64 sec	34 sec	68 sec

From the results, we notice that the performance of the “Serial + Parallel Approach” is not better than the “Serial approach” this is probably due to the overhead introduced by the creation of the process and the process management. We can see from the table that the “Parallel approach” is the best one in each of the three cases. Furthermore, we note that when the video length increases, the difference in terms of performance between "Serial approach" and "Serial + Parallel Approach" are more or less similar while the differences in performance between the two methods mentioned above and the "Parallel approach" are remarkable.

All the results showed before are performed on a personal computer with an octa-core CPU, we can see how the duration of the video affects the performances in fact the video with a higher duration shows a bigger improvement on the duration of the computation when parallelization is active.

We also tried to apply the same approach on Raspberry PI 3B+ and we obtained the following results:

VIDEO	DURATION	1 core	2 cores	3 cores	4 cores
VideoAudi.mp4	0:11	169 sec	132 sec	115 sec	118 sec