**departamento de eletrónica, telecomunicações e informática**

|  |  |
| --- | --- |
| Curso | 8309 - Mestrado Integrado em Engenharia Eletrónica e Telecomunicações |
| Disciplina | 41500 – Visão por Computadores na Indústria |
| Ano letivo | 2020/21 |

Milestone 1

Deteção de cores de Lego

Autores:

|  |  |  |  |
| --- | --- | --- | --- |
| [77848]  [89833] | [Fábio Almeida]  [Micael Ramos] | | |
| [90367] | [Ana Sousa] | | |
| Turma | [P1] | Grupo | [10] |

|  |  |
| --- | --- |
| Data | [03/05/2021] |
| Docente | [António Neves]  [Daniel Canedo] |

|  |  |
| --- | --- |
| Resumo: | O presente relatório descreve a fase inicial do projeto de desenvolvimento de um sistema, com recurso da biblioteca *OpenCv,* através da linguagem *Python* que permite detetar peças de lego e com recurso de um *Raspberry Pi 4, cuja camara já se encontra instalada na mesma*. Nesta fase inicial, pretendemos identificar as cores de cada peça do lego. Aqui, são apresentados já alguns resultados obtidos e algumas conclusões desses mesmos resultados. |
|  |  |

# Introdução

A cadeira de Visão por Computador na Indústria tem como objetivo o desenvolvimento de um projeto para identificação de cores e de formas dos legos com recurso à biblioteca *OpenCv* e à linguagem de *Python*. O projeto consiste na deteção de peças de legos em diferentes condições, de modo a ser possível o reconhecimento da cor e da dimensão.

As especificações impostas para a primeira fase do projeto foram:

* Captura de imagem e vídeo;
* Deteção e identificação de cores.

Já para conseguirmos finalizar e alcançar o objetivo do projeto, as especificações impostas foram:

* Captura e calibração da câmara do *Raspberry* em tempo real;
* Desenvolvimento de um software utilizando a linguagem *Python*;
* Testes finais e ajustes ao software desenvolvido.

Para isso, o grupo decidiu estar constantemente em comunicação e ter um lado crítico, uma vez que o projeto se insere em condições reais que por vezes dificultam os resultados esperados.

# Resultados e Análise de Resultados

Para conseguirmos obter resultados, é de notar que, para esta fase, foi utilizado o *Datasheet* de peças de Lego fornecido pelo professor. Assim, todos os algoritmos criados até então, terão de ser devidamente modificados para um *Datasheet* criado por nós, que será muito próximo do *setup* final. Deste modo, passámos então a explicar todos os resultados obtidos.

Funcionamento do programa de configuração do sistema:

Numa primeira fase, o objetivo era escolher um ponto através do rato para definir os limites inicias da máscara para uma determinada cor e, para isso, a figura 1 mostra a janela inicial com imagem que foi utilizada para atingir este objetivo.

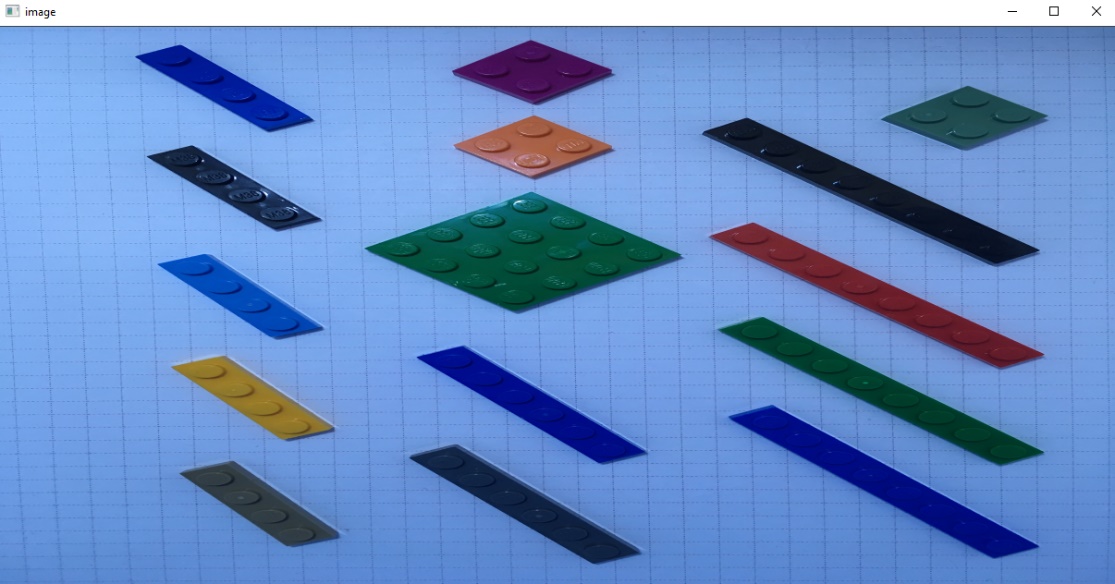


Figure 1 - Janela inicial

Após selecionar o ponto desejado o programa mostra a imagem inicial com a máscara inicial aplicada, e através de cliques nas partes ainda não incluídas na máscara (partes a preto da imagem) é possível ajustar a máscara até incluir toda a gama de cor desejada, tal como mostra a figura 2 até à 4.

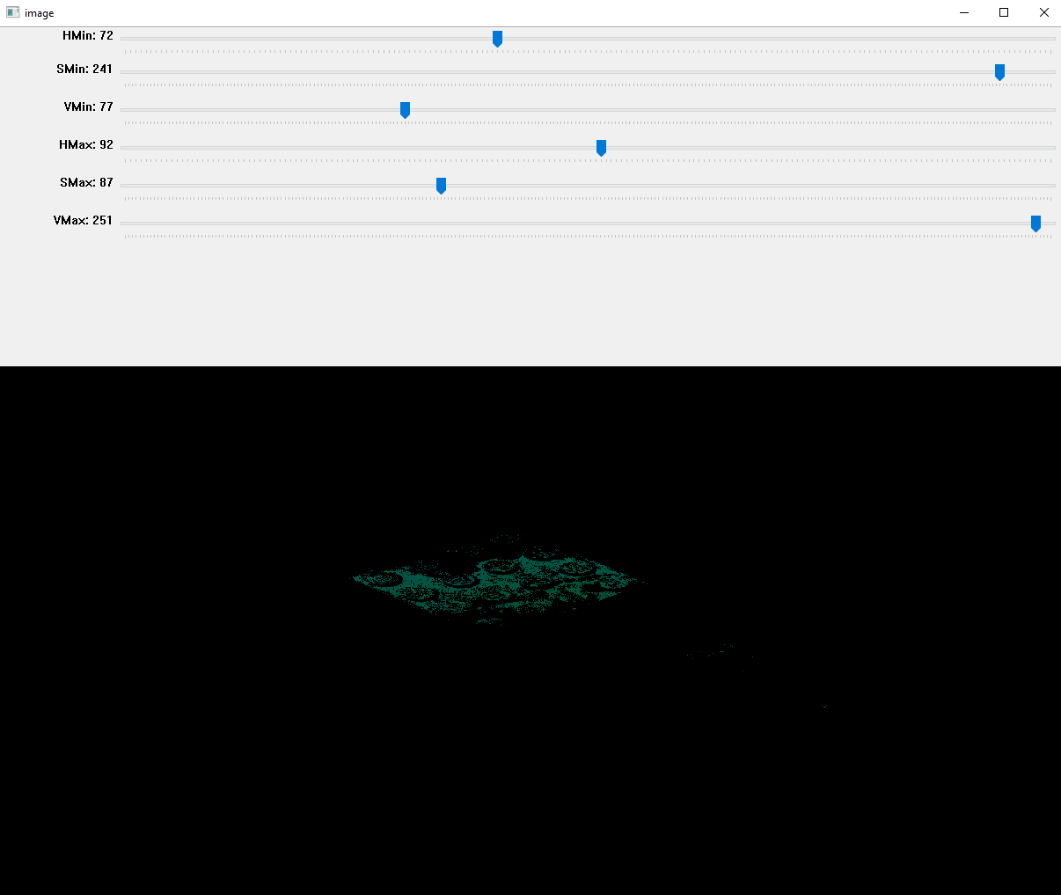
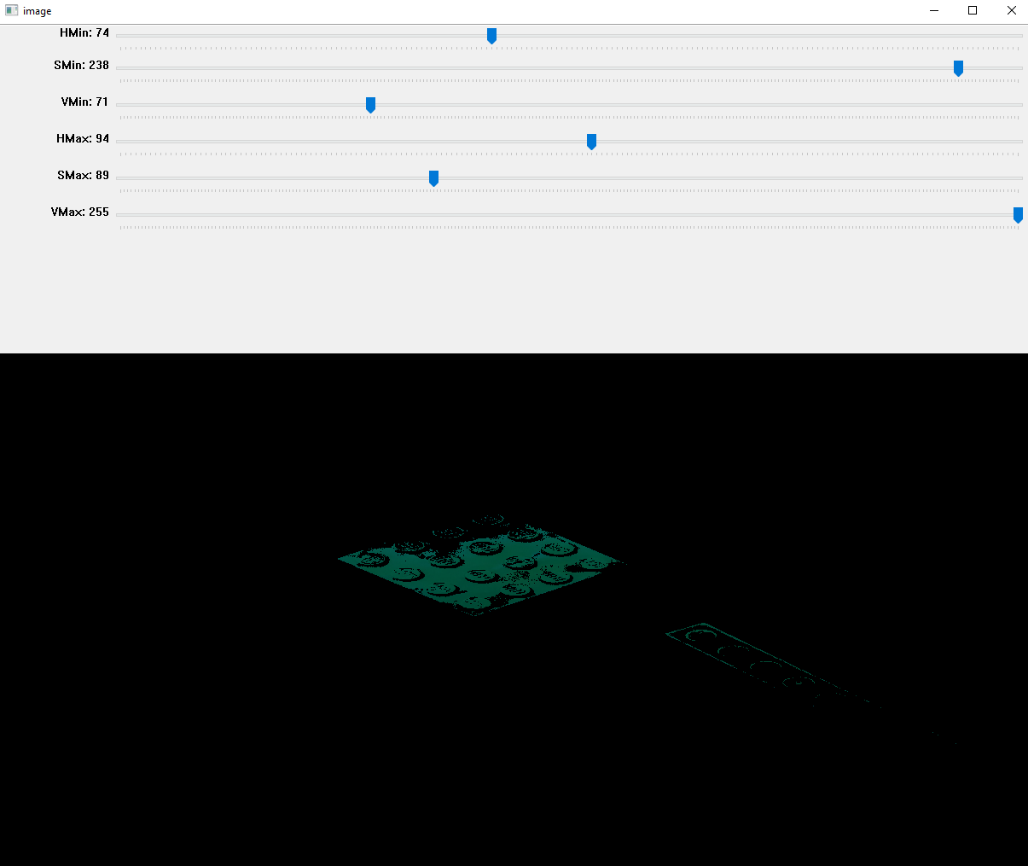


Figure 3 - Janela após alguns ajustes

Figure 2 - Janela após escolher o ponto inicial

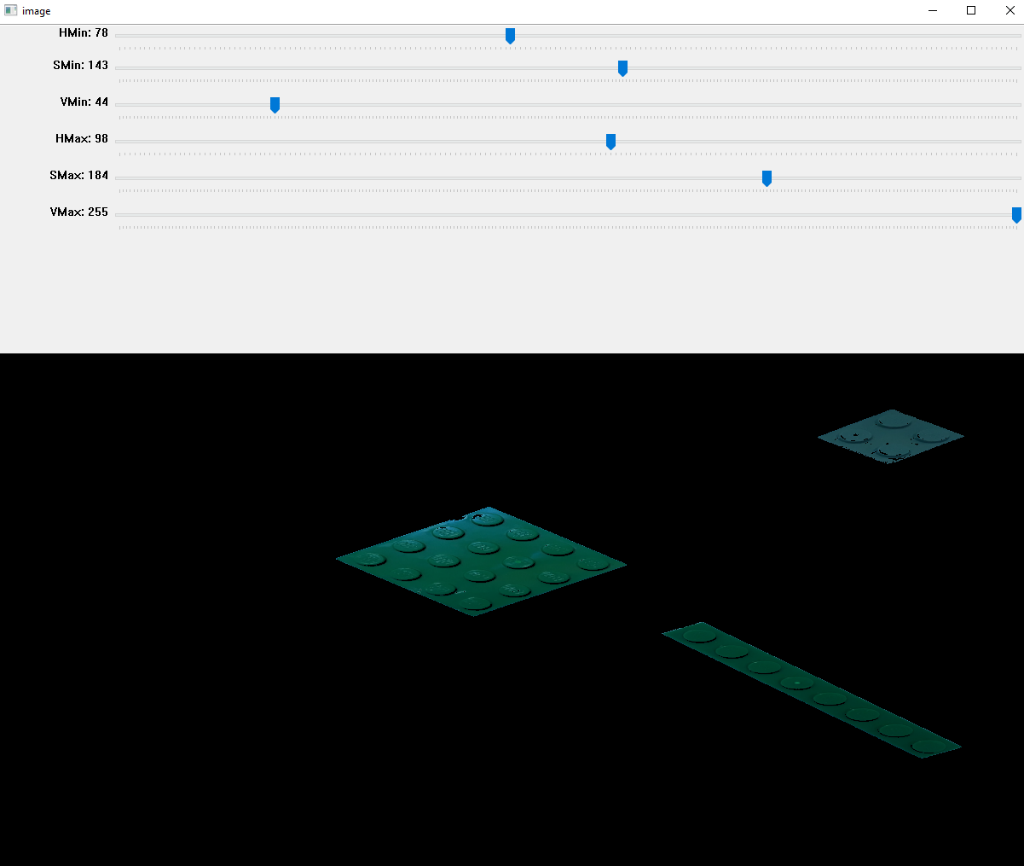


Figure 4 - Janela após todos os ajustes

Se durante o processo de ajuste da máscara for selecionada alguma parte da imagem indesejada, é possível voltar ao estado anterior ao último click.

Após ajustar a máscara até ao ponto desejado, esta é adicionada a um dicionário e após as máscaras de todas as cores desejadas estarem bem definidas e adicionadas ao dicionário este é guardado num ficheiro *.txt* para mais tarde ser usado para identificar as cores noutras imagens (Fig. 5).

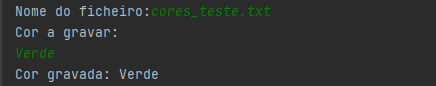


Figure 5 - Definição do nome do ficheiro e cor a ser gravada

Assim, depois de todas as cores que o sistema é capaz de identificar serem escolhidas, e guardadas num dicionário e num ficheiro, é possível agora, através de outro programa, aplicar estas máscaras a outras imagens para identificar os diferentes legos das várias cores, tal como mostra a figura abaixo (figura 6).

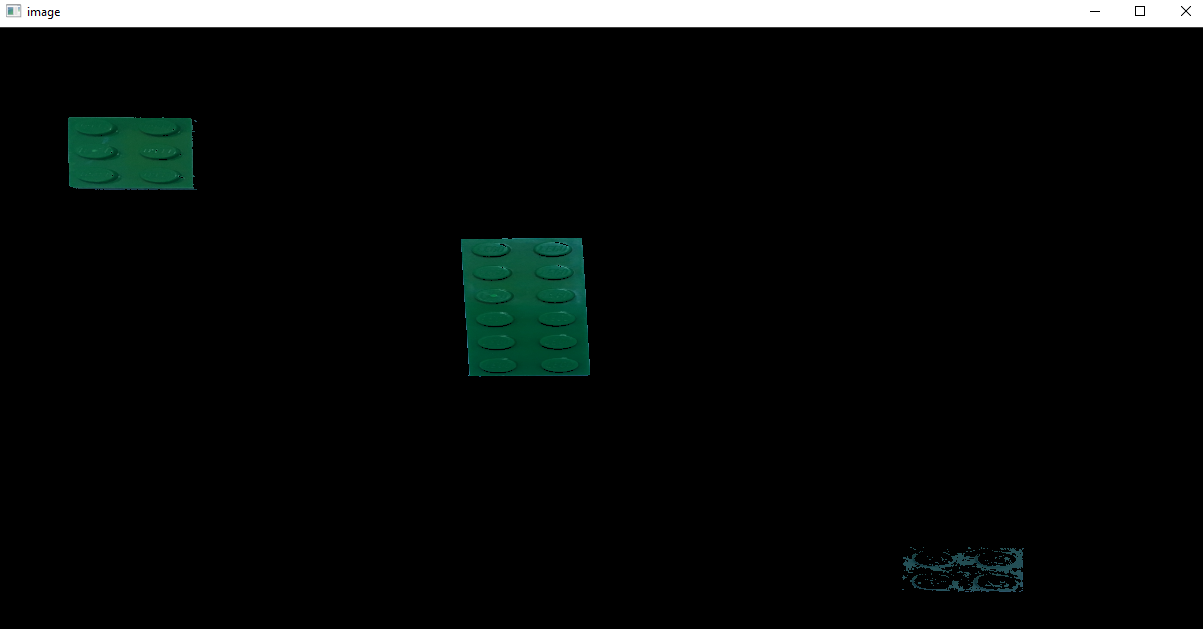


Figure 6 - Imagem resultante da aplicação das máscaras definidas



Figure 7 - Figura original

De notar que, na imagem obtida verifica-se que um dos legos não é totalmente bem identificado, isto deve-se a estarmos a usar imagens retiradas do *Datasheet* fornecido pelo Professor, onde não temos a certeza se a luz utilizada na obtenção das duas imagens (usada no programa de calibração e usada no teste) é a mesma. No nosso caso, uma das especificações do projeto é ter o controlo total da luz, isto é, todas as imagens são obtidas sob as mesmas condições de luz.

# Conclusões

No que diz respeito a esta primeira fase do projeto, os objetivos definidos no início foram em grande parte superados. No entanto, sabemos que, como estamos numa fase inicial, ainda temos muitos aspetos a melhorar, bem como o código e a identificação das peças não estarem uniformes e terem os círculos como no *datasheet* fornecido.

De modo a melhorarmos estes aspetos, o nosso próximo passo será ter em atenção todo o trabalho feito até então, e em consequência fazer certos ajustes para que possamos cumprir todos os objetivos como pretendemos e com sucesso. Depois de concluído este passo, iremos dar continuação e terminar a tarefa da deteção dos contornos das peças do *datasheet*.

# Anexos

**def** mouse\_click():  
 **import** cv2  
 **global** init  
  
 *# Variables* init = [0,0,0]  
  
 *# Load image* img = cv2.imread(cv2.samples.findFile(**"legos3.png"**))  
  
 *# Reduce the image* height, width = img.shape[:2]  
 size = (int(width \* 0.4), int(height \* 0.15)) *# bgr* img = cv2.resize(img, size, interpolation=cv2.INTER\_AREA)  
  
 *# Convert BGR to HSV* HSV = cv2.cvtColor(img, cv2.COLOR\_BGR2HSV)  
  
 *# Mouse left button action* **def** getposHsv(event, x, y, flags, param):  
 **global** init  
 **if** event == cv2.EVENT\_LBUTTONDOWN:  
 print(**"HSV is"**, HSV[y, x])  
 init[0]=HSV[y,x][0]  
 init[1] = HSV[y, x][1]  
 init[2] = HSV[y, x][2]  
  
 *# Display the result image* cv2.imshow(**'image'**, img)  
  
 *# Call mouse left button action* cv2.setMouseCallback(**"image"**, getposHsv)  
  
 **while**(1):  
 k = cv2.waitKey(20) & 0xFF  
 **if** k == 27:  
 **break  
 elif** k == ord(**'s'**):  
 print(**'init is'**,init)  
 **return** init  
 cv2.destroyAllWindows()

*# min\_max.py  
#  
# System Calibration program  
# This program allows us to creat all the mask for the color that the system can identify  
#  
# Running process:  
# - Pick a point with the left mouse button  
# - Press 's' to save the point picked  
# / A new window opens  
# - Click in a point of the image to include it in the mask (if click in a point and realize that you don't want the result, you can press 'r' and the mask return to the previous state)  
# - After the mask is fine press 's' to save it  
# - Write the name of the file where the masks will be saved  
# - Write the color of the maks created  
# - Check the message with the indication that the color has been recorded  
# - Repeat the process for all desired colors***import** cv2  
**import** numpy **as** np  
**import** ast  
**from** mouse **import** mouse\_click  
  
*# Variables***global** init  
init = [0,0,0]  
cores = {}  
  
*# Call mouse\_click() function in order to obtaining the initial limit values of the mask*init\_val = mouse\_click()  
  
**def** nothing(x):  
 **pass***# Load image*image = cv2.imread(cv2.samples.findFile(**"legos3.png"**))  
  
*# Mouse right button action***def** getposHsv\_right(event, x, y, flags, param):  
 **global** hMin,sMin,vMin,hMax,sMax,vMax,hMin\_p,sMin\_p,vMin\_p,hMax\_p,sMax\_p,vMax\_p  
 **if** event == cv2.EVENT\_RBUTTONDOWN:  
 print(**"HSV is"**, hsv[y, x])  
  
 *# Save the current values before update to allows us to return to previus mask state* hMin\_p = hMin  
 sMin\_p = sMin  
 vMin\_p = vMin  
 hMax\_p = hMax  
 sMax\_p = sMax  
 vMax\_p = vMax  
  
 *# See if the HSV values of the picked pixel is outside the limits of the mask, and if so, update them* **if**(hsv[y, x][0] < hMin):  
 hMin=hsv[y, x][0]  
 **if**(hsv[y, x][1] < sMin):  
 sMin = hsv[y, x][1]  
 **if**(hsv[y, x][2] < vMin):  
 vMin = hsv[y, x][2]  
 **if**(hsv[y, x][0] > hMax):  
 hMax = hsv[y, x][0]  
 **if**(hsv[y, x][1] > sMax):  
 sMax = hsv[y, x][1]  
 **if**(hsv[y, x][2] > vMax):  
 vMax = hsv[y, x][2]  
  
  
  
*# Reduce the image*height, width = image.shape[:2]  
size = (int(width \* 0.4), int(height \* 0.15)) *# bgr*image = cv2.resize(image, size, interpolation=cv2.INTER\_AREA)  
  
*# Create a window*cv2.namedWindow(**'image'**)  
  
*# Create the trackbars for see the color change*cv2.createTrackbar(**'HMin'**, **'image'**, 0, 179, nothing)  
cv2.createTrackbar(**'SMin'**, **'image'**, 0, 255, nothing)  
cv2.createTrackbar(**'VMin'**, **'image'**, 0, 255, nothing)  
cv2.createTrackbar(**'HMax'**, **'image'**, 0, 179, nothing)  
cv2.createTrackbar(**'SMax'**, **'image'**, 0, 255, nothing)  
cv2.createTrackbar(**'VMax'**, **'image'**, 0, 255, nothing)  
  
  
*# Initialize HSV min/max values with the returned by the mouse\_click function*hMin = init\_val[0]-10  
sMin = init\_val[1]-5  
vMin = init\_val[2]-5  
hMax = init\_val[0]+10  
sMax = init\_val[1]+5  
vMax = init\_val[2]+5  
  
  
**while**(1):  
  
 *# Call mouse right button action* cv2.setMouseCallback(**"image"**, getposHsv\_right)  
  
 *# Set the limit values of the mask in to trackbars* cv2.setTrackbarPos(**'HMin'**, **'image'**, hMin)  
 cv2.setTrackbarPos(**'SMin'**, **'image'**, sMin)  
 cv2.setTrackbarPos(**'VMin'**, **'image'**, vMin)  
 cv2.setTrackbarPos(**'HMax'**, **'image'**, hMax)  
 cv2.setTrackbarPos(**'VMax'**, **'image'**, sMax)  
 cv2.setTrackbarPos(**'SMax'**, **'image'**, vMax)  
  
 *# Set minimum and maximum HSV values to display* lower = np.array([int(hMin), int(sMin), int(vMin)])  
 upper = np.array([int(hMax), int(sMax), int(vMax)])  
 hsv = cv2.cvtColor(image, cv2.COLOR\_BGR2HSV)  
  
 *# Create the mak* mask = cv2.inRange(hsv, lower, upper)  
  
 *# Create an image using the mask* result = cv2.bitwise\_and(image, image, mask=mask)  
  
 *# Display the result image* cv2.imshow(**'image'**, result)  
 k = cv2.waitKey(10) & 0xFF  
 **if** k == ord(**'q'**):  
 **break** *# if you press 'r' the mask return to the previous state* **elif** k == ord(**'r'**):  
 print(hMin\_p,sMin\_p,vMin\_p,hMax\_p,sMax\_p,vMax\_p)  
 hMin = hMin\_p  
 sMin = sMin\_p  
 vMin = vMin\_p  
 hMax = hMax\_p  
 sMax = sMax\_p  
 vMax = vMax\_p  
  
 *# if maks is fine and you wont to save it* **elif** k == ord(**'s'**):  
  
 *# read the name of the file, open it and read the dictionary* **try**:  
 nome\_arquivo = input(**'Nome do ficheiro:'**)  
 arquivo = open(nome\_arquivo, **'r+'**)  
 contente = arquivo.read()  
 cores = ast.literal\_eval(contente)  
 arquivo.seek(0)  
 arquivo.truncate(0)  
 *# if file does not exist, create it* **except** FileNotFoundError:  
 arquivo = open(nome\_arquivo, **'w+'**)  
  
  
  
 print(**"Cor a gravar:"**)  
 color = input()  
  
 *# see if the color already exists* **if**(str(color+**'\_l'**) **in** cores):  
 print(**'Cor já existe'**)  
 **break  
 else**:  
 texto\_l = (str(hMin) + **','** + str(sMin) + **','** + str(vMin))  
 texto\_u = (str(hMax) + **','** + str(sMax) + **','** + str(vMax))  
 cores[color+**'\_l'**] = texto\_l  
 cores[color+**'\_u'**] = texto\_u  
  
 *# write the dictionary in to the file* arquivo.write(str(cores))  
 arquivo.close()  
 print(**"Cor gravada:"**, color)  
  
 *# call the mouse\_click function again to repeat the all the process for every color you want* init\_val=mouse\_click()  
 hMin = init\_val[0] - 10  
 sMin = init\_val[1] - 5  
 vMin = init\_val[2] - 5  
 hMax = init\_val[0] + 10  
 sMax = init\_val[1] + 5  
 vMax = init\_val[2] + 5  
  
cv2.destroyAllWindows()

**if**(hsv[y, x][0] > hMax):  
 hMax = hsv[y, x][0]  
 **if**(hsv[y, x][1] > sMax):  
 sMax = hsv[y, x][1]  
 **if**(hsv[y, x][2] > vMax):  
 vMax = hsv[y, x][2]  
  
  
  
*# Reduce the image*height, width = image.shape[:2]  
size = (int(width \* 0.4), int(height \* 0.15)) *# bgr*image = cv2.resize(image, size, interpolation=cv2.INTER\_AREA)  
  
*# Create a window*cv2.namedWindow(**'image'**)  
  
*# Create the trackbars for see the color change*cv2.createTrackbar(**'HMin'**, **'image'**, 0, 179, nothing)  
cv2.createTrackbar(**'SMin'**, **'image'**, 0, 255, nothing)  
cv2.createTrackbar(**'VMin'**, **'image'**, 0, 255, nothing)  
cv2.createTrackbar(**'HMax'**, **'image'**, 0, 179, nothing)  
cv2.createTrackbar(**'SMax'**, **'image'**, 0, 255, nothing)  
cv2.createTrackbar(**'VMax'**, **'image'**, 0, 255, nothing)  
  
  
*# Initialize HSV min/max values with the returned by the mouse\_click function*hMin = init\_val[0]-10  
sMin = init\_val[1]-5  
vMin = init\_val[2]-5  
hMax = init\_val[0]+10  
sMax = init\_val[1]+5  
vMax = init\_val[2]+5  
  
  
**while**(1):  
  
 *# Call mouse right button action* cv2.setMouseCallback(**"image"**, getposHsv\_right)  
  
 *# Set the limit values of the mask in to trackbars* cv2.setTrackbarPos(**'HMin'**, **'image'**, hMin)  
 cv2.setTrackbarPos(**'SMin'**, **'image'**, sMin)  
 cv2.setTrackbarPos(**'VMin'**, **'image'**, vMin)  
 cv2.setTrackbarPos(**'HMax'**, **'image'**, hMax)  
 cv2.setTrackbarPos(**'VMax'**, **'image'**, sMax)  
 cv2.setTrackbarPos(**'SMax'**, **'image'**, vMax)  
  
 *# Set minimum and maximum HSV values to display* lower = np.array([int(hMin), int(sMin), int(vMin)])  
 upper = np.array([int(hMax), int(sMax), int(vMax)])  
 hsv = cv2.cvtColor(image, cv2.COLOR\_BGR2HSV)  
  
 *# Create the mak* mask = cv2.inRange(hsv, lower, upper)  
  
 *# Create an image using the mask* result = cv2.bitwise\_and(image, image, mask=mask)  
  
 *# Display the result image* cv2.imshow(**'image'**, result)  
 k = cv2.waitKey(10) & 0xFF  
 **if** k == ord(**'q'**):  
 **break** *# if you press 'r' the mask return to the previous state* **elif** k == ord(**'r'**):  
 print(hMin\_p,sMin\_p,vMin\_p,hMax\_p,sMax\_p,vMax\_p)  
 hMin = hMin\_p  
 sMin = sMin\_p  
 vMin = vMin\_p  
 hMax = hMax\_p  
 sMax = sMax\_p  
 vMax = vMax\_p  
  
 *# if maks is fine and you wont to save it* **elif** k == ord(**'s'**):  
  
 *# read the name of the file, open it and read the dictionary* **try**:  
 nome\_arquivo = input(**'Nome do ficheiro:'**)  
 arquivo = open(nome\_arquivo, **'r+'**)  
 contente = arquivo.read()  
 cores = ast.literal\_eval(contente)  
 arquivo.seek(0)  
 arquivo.truncate(0)  
 *# if file does not exist, create it* **except** FileNotFoundError:  
 arquivo = open(nome\_arquivo, **'w+'**)  
  
  
  
 print(**"Cor a gravar:"**)  
 color = input()  
  
 *# see if the color already exists* **if**(str(color+**'\_l'**) **in** cores):  
 print(**'Cor já existe'**)  
 **break  
 else**:  
 texto\_l = (str(hMin) + **','** + str(sMin) + **','** + str(vMin))  
 texto\_u = (str(hMax) + **','** + str(sMax) + **','** + str(vMax))  
 cores[color+**'\_l'**] = texto\_l  
 cores[color+**'\_u'**] = texto\_u  
  
 *# write the dictionary in to the file* arquivo.write(str(cores))  
 arquivo.close()  
 print(**"Cor gravada:"**, color)  
  
 *# call the mouse\_click function again to repeat the all the process for every color you want* init\_val=mouse\_click()  
 hMin = init\_val[0] - 10  
 sMin = init\_val[1] - 5  
 vMin = init\_val[2] - 5  
 hMax = init\_val[0] + 10  
 sMax = init\_val[1] + 5  
 vMax = init\_val[2] + 5

*# Display the result image* cv2.imshow(**'image'**, result)  
 k = cv2.waitKey(10) & 0xFF  
 **if** k == ord(**'q'**):  
 **break** *# if you press 'r' the mask return to the previous state* **elif** k == ord(**'r'**):  
 print(hMin\_p,sMin\_p,vMin\_p,hMax\_p,sMax\_p,vMax\_p)  
 hMin = hMin\_p  
 sMin = sMin\_p  
 vMin = vMin\_p  
 hMax = hMax\_p  
 sMax = sMax\_p  
 vMax = vMax\_p  
  
 *# if maks is fine and you wont to save it* **elif** k == ord(**'s'**):  
  
 *# read the name of the file, open it and read the dictionary* **try**:  
 nome\_arquivo = input(**'Nome do ficheiro:'**)  
 arquivo = open(nome\_arquivo, **'r+'**)  
 contente = arquivo.read()  
 cores = ast.literal\_eval(contente)  
 arquivo.seek(0)  
 arquivo.truncate(0)  
 *# if file does not exist, create it* **except** FileNotFoundError:  
 arquivo = open(nome\_arquivo, **'w+'**)  
  
  
  
 print(**"Cor a gravar:"**)  
 color = input()  
  
 *# see if the color already exists* **if**(str(color+**'\_l'**) **in** cores):  
 print(**'Cor já existe'**)  
 **break  
 else**:  
 texto\_l = (str(hMin) + **','** + str(sMin) + **','** + str(vMin))  
 texto\_u = (str(hMax) + **','** + str(sMax) + **','** + str(vMax))  
 cores[color+**'\_l'**] = texto\_l  
 cores[color+**'\_u'**] = texto\_u  
  
 *# write the dictionary in to the file* arquivo.write(str(cores))  
 arquivo.close()  
 print(**"Cor gravada:"**, color)  
  
 *# call the mouse\_click function again to repeat the all the process for every color you want* init\_val=mouse\_click()  
 hMin = init\_val[0] - 10  
 sMin = init\_val[1] - 5  
 vMin = init\_val[2] - 5  
 hMax = init\_val[0] + 10  
 sMax = init\_val[1] + 5  
 vMax = init\_val[2] + 5  
  
cv2.destroyAllWindows()

*# mask.py  
#  
# Program to apply maks  
# This program allows us to select the a color and apply that color mask  
#  
# Running process:  
# - Write the name of the file where the masks are saved  
# - Write the desired color  
# - Press 'q' to finish the program***import** cv2  
**import** numpy **as** np  
**import** ast  
  
*# Load image*image = cv2.imread(cv2.samples.findFile(**"legos4.png"**))  
  
*# Reduce the image*height, width = image.shape[:2]  
size = (int(width \* 0.4), int(height \* 0.15)) *# bgr*image = cv2.resize(image, size, interpolation=cv2.INTER\_AREA)  
  
*# Open file*nome\_arquivo = input(**'Nome do ficheiro:'**)  
arquivo = open(nome\_arquivo, **'r+'**)  
contente = arquivo.read()  
  
*# If file is empty close program***if**(contente == **''**):  
 print(**'Ficheiro nao contem nenhuma cor'**)  
 cv2.destroyAllWindows()  
 quit()  
  
*# Import Dictionary*cores = ast.literal\_eval(contente)  
  
*# Read desired color*cor = input(**'Cor:'**)  
teste = str(cor+**'\_l'**)  
  
*# If color exist in the dictionary continue***if**(teste **in** cores):  
  
 string\_l = str(cores[cor + **'\_l'**])  
 string\_u = str(cores[cor + **'\_u'**])  
  
 lower = np.fromstring(string\_l, sep=**','**)  
 upper = np.fromstring(string\_u, sep=**','**)  
  
 hsv = cv2.cvtColor(image, cv2.COLOR\_BGR2HSV)  
  
 *# Create mask* mask = cv2.inRange(hsv, lower, upper)  
 result = cv2.bitwise\_and(image, image, mask=mask)  
  
 **while** (1):  
  
 *# Display result image* cv2.imshow(**'image'**, result)  
 k = cv2.waitKey(10) & 0xFF  
 **if** k == ord(**'q'**):  
 **break  
else**:  
 print(**'Cor nao existe'**)  
  
cv2.destroyAllWindows()