Report Engineering Programming

Abdessamad Nassihi,Ryan Chamhi
2nd Bachelor Industrial Engineering

Summary

1.0 Introduction

The project of engineering programming consists of making two applications: an image manipulation tool and a covid data visualization tool.

The image manipulation app should have a certain number of manipulation features: a RGB and gamma modification feature, an image shape modification feature and a filter application. The user should be able to choose an image from the disk, apply some chosen modifications to the image and save the modified image. The user should olso directly see the modifications he is making.

The covid data visualization tool should read data from a CSV file and show this data using plots. The user should be able to choose which country he wants to visualize the data of. The app should also make a comparison between multiple countries possible.

With this project, we will be able to put into practice the things we learned during the semester.

2.0 Image Manipulation Application

2.1 Features of the appliaction

With the image manipulation application the user will be able to choose an image from the disk, perform some manipulations and at the end saves the manipulated image. The user will be able to :

- Modify the value of the red, green and blue channels of the image, as well as its gamma
- Modify the shape of the image
- Mirror the image (horizontally or vertically)
- Add filters

2.2 Useful functions

2.2.1 Library

To make certain manipulations possible we have to begin with adding some libraries. We will use tkinter for our GUI.

In [ ]:

```
import tkinter as tk
from tkinter import ttk
from tkinter import filedialog
from tkinter import messagebox
from PIL import Image, ImageTk
import matplotlib.pylab as plt
import numpy as np
```

2.2.2 Open image from disk

To let the user choose an image from the disk we imported filedialog from tkinter. The method filedialog.askopenfilename that takes the required types as a parameter, will return the pad of the image the user chooses. By assigning a variable to the pad of the image we can open and show the image on the GUI.

In [ ]:

```
f_types = [('PNG files', '*.png'),('Jpg Files','*.jpg')] # files types
# access image from disk and returns the pad
filename = filedialog.askopenfilename(filetypes=f_types)
```

2.2.3 Import and load the image

For this feature we use the matplotlib library, with the method imread(). This method returns the image data as a multidimensional numpy array of shape (M,N,3). M describes the number of rows, N is the number of columns and 3 is the size of the vector which represents the RGB value.

In [ ]:

```
data = plt.imread(filename)
```

Example:

In [11]:

```
import matplotlib.pylab as plt
import numpy as np
data = plt.imread('lenna.png') # 'city.png','animal.png','monalisa.png'
plt.imshow(data)
plt.axis('off')
```

Out[11]:

```
(-0.5, 511.5, 511.5, -0.5)
```

2.2.4 Image plotting

To plot the image on the GUI in the form of a label we first create the image from the numpy array containing the data of the manipulated image. The method ImageTk.PhotoImage() from the pillow library makes this possible. We then convert the manipulated image to a Tkinter-compatible photo image using the ImageTk.PhotoImage() also from the pillow library.

In [ ]:

```
ManImg = Image.fromarray((arr * 255).astype(np.uint8))
ManImage = ImageTk.PhotoImage(ManImage)
# Image label propreties
ManipulatedImage = tk.Label(root,image=ManImage,text = 'Manipulated image',
                            font = ("Courrier",25,'bold'),
                                compound='bottom',bg ='#811ef8',fg = 'white')
ManipulatedImage.place(x=850,y=150)
# add label to the root
```

2.2.5 Saving the image

In order to save the manipulated image in the disk we use the Image.save() method from the pillow library.

In [ ]:

```
ManImg.save('ManipulatedImage.png') # Saves manipulated image to disk
```

2.3 Modification features

2.3.1 RGB and gamma modification feature

The function takes the channel to modify and the sign(increase or decrease the value of the channel or gamma). To be able to manipulate the values of the RGB channels we use the numpy array slicing capabilities. We will apply the gamma modification with the exponential: pixel**gamma

In [ ]:

```
fig, axs = plt.subplots(nrows=1, ncols=4, figsize=(10,10))
def ModifyRGB(data,val,color): # function for the RGB manipulation
    val = val/255.0
    data[:,:,color] += val
    axs[color].imshow(data)
    axs[color].axis('off')
def ModifyGamma(data,val):
    NewData = data ** (1/val)
    axs[3].imshow(NewData)
    axs[3].axis('off')
```

In [4]:

```
for i in range(3):
    ModifyRGB(plt.imread('animal.png'),25,i)
    ModifyGamma(plt.imread('animal.png'),0.3)
```

2.3.2 Shape modification feature

The function takes 4 values: how much to keep (or cut) of each side of the image. The four values are given by the user. We use the slicing capabilities of numpy arrays to be able to cut the image.

In [ ]:

```
fig, axs = plt.subplots(nrows=1, ncols=2, figsize=(10,10))
def CropImage(data,left,right,top,bottom):
    axs[0].imshow(data)
    axs[0].axis('off')
    if (right == 0 and bottom ==0):
        CroppedImg = data[top:,left:]
    elif bottom == 0:
        CroppedImg = data[top:,left:-right]
    elif right == 0:
        CroppedImg = data[top:-bottom,left:]
    else:
        CroppedImg = data[top:-bottom,left:-right]
    axs[1].imshow(CroppedImg)
    axs[1].axis('off')
```

In [12]:

```
CropImage(plt.imread('monalisa.png'),20,50,90,90)
```

2.3.3 Mirror feature

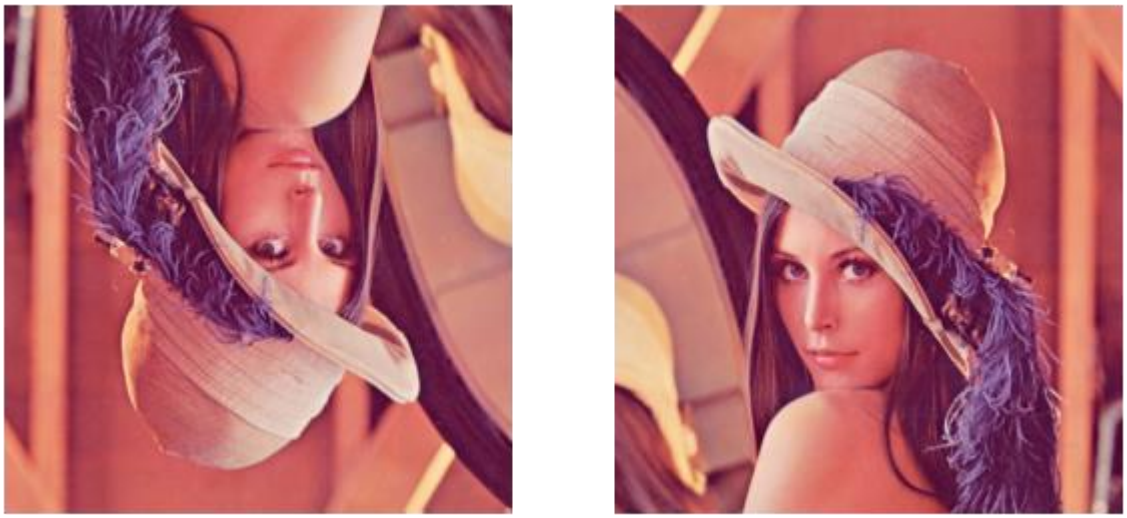
The function will flip an image given the axis as a parameter. The user can choose if the image is mirrored vertically (meaning from left to right, or right to left), or horizontally. We can mirror the image by inverting the rows or columns using the slicing feature: `[::-1]`

```
fig, axs = plt.subplots(nrows=1, ncols=2, figsize=(10,10))
def MirrorImage(data):
    MirrorHor = data[::-1,:,:]
    axs[0].imshow(MirrorHor)
    axs[0].axis('off')

    MirrorVer = data[:,::-1,:]
    axs[1].imshow(MirrorVer)
    axs[1].axis('off')
```

In [14]:

```
MirrorImage(plt.imread('lenna.png'))
```



2.3.4 Sepia filter

To create a sepia filter we modify the RGB values of the image using the following formula:

- newR = 0.393*r* + *0.769*g + 0.189*b
- newG = 0.349*r* + *0.686*g + 0.168*b
- newB = 0.272*r* + *0.534*g + 0.131*b

We slice the numpy array for accessing the RGB values of each pixel. We then keep the RGB values between 0 and 1

```
fig, axs = plt.subplots(nrows=1, ncols=2, figsize=(10,10))
def sepiaFilter(data):
    Sepia_Filter = data.copy()
    newR = 0.393*data[:,:,0] + 0.769*data[:,:,1] + 0.189*data[:,:,2]
    newG = 0.349*data[:,:,0] + 0.686*data[:,:,1] + 0.168*data[:,:,2]
    newB = 0.272*data[:,:,0] + 0.534*data[:,:,1] + 0.131*data[:,:,2]

    for i in range(512):
        for j in range(512):
            colors = [newR[i][j],newG[i][j],newB[i][j]]
            Sepia_Filter[i][j] = colors

    Sepia_Filter[Sepia_Filter > 1] = 1

    axs[0].imshow(data)
    axs[0].axis("off")

    axs[1].imshow(Sepia_Filter)
    axs[1].axis("off")
```

In [34]:

```
sepiaFilter(plt.imread('lenna.png'))
```

2.3.5 Pixelation

To create a pixelated image we first have to divide the image into k×k squares, with K chosen by the user. We then replace all the pixels in each block with the average value of the pixels in that block. We repeat this operation for each block that composed the image.

In this part we will implement two codes, a version in plain python and a version using numpy's array manipulation capabilities. We will then compare the performances.

Plain python version:

```
import matplotlib.pylab as plt
import numpy as np
fig, axs = plt.subplots(nrows=1, ncols=2, figsize=(10,10))
def Pixelation1(data,k):
    rows,cols = (data.shape[0],data.shape[1])
    rows, cols = rows - rows % k, cols - cols % k
    PixelatedImg = []
    for i in range(cols):
        PixelatedImg.append([])
        for j in range(rows):
            PixelatedImg[i].append([])
            for l in range(3):
                 PixelatedImg[i][j].append(0) # creating a black image
    SomR =0
    SomG =0
    SomB =0
    arr =[]
    for x in range(0,rows,k):
        for y in range(0,cols,k):
            for i in range (x,x+k):
                for j in range(y,y+k):
                    SomR += data[i,j,0]
                    SomG += data[i,j,1]
                    SomB += data[i,j,2]
            avR = SomR/(k*k)
            avG = SomG/(k*k)
            avB = SomB/(k*k)
            colors = [avR,avG,avB]
            SomR =0
            SomG =0
            SomB =0
            arr.append(colors)
    n=0
    for x in range(0,rows,k):
        for y in range(0,cols,k):
            n+=1
            for i in range (x,(x+k)):
                for j in range(y,(y+k)):
                    PixelatedImg[i][j] = arr[n-1]
    axs[0].imshow(data)
    axs[0].axis('off')

    axs[1].imshow(PixelatedImg)
    axs[1].axis('off')
```

Showing the performance:

In [20]:

```
%timeit version1 = Pixelation1(plt.imread('lenna.png'),17)
```

```
473 ms ± 3.53 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

Version using numpy's array:

In [ ]:

```
fig, axs = plt.subplots(nrows=1, ncols=2, figsize=(10,10))
def Pixelation2(data,k):
    rows,cols = (data.shape[0],data.shape[1])
    rows, cols = rows - rows % k, cols - cols % k
    PixelatedImg = np.zeros((rows, cols, 3))
    for x in range(0, rows, k):
        for y in range(0, cols, k):
            PixelatedImg[x:x+k,y:y+k] = data[x:x+k,y:y+k].mean(axis=(0,1))
    axs[0].imshow(data)
    axs[0].axis('off')

    axs[1].imshow(PixelatedImg)
    axs[1].axis('off')
```

Showing the performance:

In [23]:

```
%timeit version2 = Pixelation2(plt.imread('lenna.png'),17)
```

```
32.4 ms ± 214 µs per loop (mean ± std. dev. of 7 runs, 10 loops each)
```

We can cleary see that the version with numpy's array runs much faster than the version in plain python.

2.0.1 Convolution

We proceed the convolution operation on the image as follows:

- We first place the kernel on the top left corner of the image
- We then multiply the corresponding pixels and take the sum of the individual results
- We repeat this operation on all the pixels of the image by each time shifting the kernel down by 1 pixel
- Finally we keep the RGB values between 0 and 1

The kernel is given by the user. Notice that each kernel creates a proper manipulation of the image.

In [ ]:

```
fig, axs = plt.subplots(nrows=1, ncols=2, figsize=(20,20))
def Convolve(data,kernel):
    row,col = (data.shape[0],data.shape[1])
    axs[0].imshow(data)
    axs[0].axis('off')
    if kernel == [[-1,-1,-1], [-1,8,-1], [-1,-1,-1]]:
        data[:] = (np.max(data,axis=-1,keepdims=1) + np.min(data,axis=-1,keepdims=1)) /2
    ConvImg = np.zeros((row+2,col+2,3))
    ConvImg[1:-1,1:-1] = data[:]
    axs[1].imshow(ConvImg)
    axs[1].axis("off")

    x = len(kernel[0])
    for i in range (0,row-1):
        for j in range (0,col-1):
            R = np.sum(kernel*ConvImg[i:i+3,j:j+3,0])
            G = np.sum(kernel*ConvImg[i:i+3,j:j+3,1])
            B = np.sum(kernel*ConvImg[i:i+3,j:j+3,2])
            data[i+1,j+1] = [R,G,B]
    data[data>1] = 1
    data[data<0] = 0
    axs[1].imshow(data)
    axs[1].axis("off")
```

In [29]:

```
Convolve(plt.imread('city.png'),[[-1,-1,-1], [-1,8,-1], [-1,-1,-1]]) # edge detection
```

In [30]:

```
Convolve(plt.imread('animal.png'),[[1/9.0,1/9.0,1/9.0],
                                   [1/9.0,1/9.0,1/9.0], [1/9.0,1/9.0,1/9.0]]) # blur effect
```

In [31]:

```
Convolve(plt.imread('monalisa.png'),[[0,-1,0], [-1,5,-1], [0,-1,0]]) # sharpen effect
```

2.0.1 Grayscale

For turning an image into grayscale we simply have to average the values of each pixel that compose the image.

In [ ]:

```
def Grayscale(data):
    data[:] = (np.max(data,axis=-1,keepdims=1) + np.min(data,axis=-1,keepdims=1)) /2
    plt.imshow(data)
    plt.axis("off")
```

In [33]:

```
Grayscale(plt.imread('city.png'))
```

3.0 Covid Data Visualization

3.1 Features of the visualization application

This application introduces the visualization and manipulation of data. With this application a user is capable of choosing the country he wants the visualize the data of and select the metrics he is interested in.

the user will be able to choose between the following metrics:

- Total cases
- Total cases per million
- Total deaths
- Hospital patients
- People vaccinated

3.2 Overview of the code

The code proceeds as follows:

- reading the csv file containing the data
- Select the desired data
- Plotting the data

3.2.1 Library

We use the pandas library to able to visualize the data. Thanks to the `matplotlib.backends.backend_tkagg` library we will plot the data on graphs that will be converted into images.

In [ ]:

```
import tkinter as tk
from tkinter import ttk
from tkinter import *
from tkinter import messagebox
import numpy as np
import pandas as pd
import matplotlib.pylab as plt
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
```

3.2.2 Reading the csv file

Storing the csv's into variabels as a dataframe with pd.read_csv():

In [13]:

```
FullData = pd.read_csv('owid-covid-data.csv')
```

3.2.3 Selecting the desired data

Metric selection

In [14]:

```
FullData = FullData.set_index('date')
Data = FullData[['location','total_cases','total_cases_per_million','total_deaths',
                 'hosp_patients','people_vaccinated']]
```

In [15]:

```
Data:
```

Out[15]:

date	location	total_cases	total_cases_per_million	total_deaths	hosp_patients	people_vac
2020-02-24	Afghanistan	5.0	0.126	NaN	NaN	
2020-02-25	Afghanistan	5.0	0.126	NaN	NaN	
2020-02-26	Afghanistan	5.0	0.126	NaN	NaN	
2020-02-27	Afghanistan	5.0	0.126	NaN	NaN	
2020-02-28	Afghanistan	5.0	0.126	NaN	NaN	
...	...	...	...	...	...	
2021-11-26	Zimbabwe	133836.0	8867.909	4704.0	NaN	375
2021-11-27	Zimbabwe	133836.0	8867.909	4704.0	NaN	376
2021-11-28	Zimbabwe	133951.0	8875.529	4705.0	NaN	377
2021-11-29	Zimbabwe	134226.0	8893.750	4706.0	NaN	378
2021-11-30	Zimbabwe	134625.0	8920.188	4707.0	NaN	

136931 rows × 6 columns

Date selection

We will only visualize the data on specific dates.

In [16]:

```
Dates = np.array(['2020-03-05','2020-04-10','2020-05-15','2020-06-20','2020-07-25',
                  '2020-08-05','2020-09-10','2020-10-15','2020-11-20','2020-12-25',
                  '2021-01-05','2021-02-10','2021-03-15','2021-04-20','2021-05-25',
                  '2021-06-05','2021-07-10','2021-08-15','2021-09-20','2021-10-25',
                  '2021-11-05'])  # desired dates
Data = Data.loc[Dates]
```

In [17]:

```
Data:
```

Out[17]:

date	location	total_cases	total_cases_per_million	total_deaths	hosp_patients	people_vac
2020-03-05	Afghanistan	5.0	0.126	NaN	NaN	
2020-03-05	Africa	24.0	0.017	NaN	NaN	
2020-03-05	Albania	NaN	NaN	NaN	NaN	
2020-03-05	Algeria	12.0	0.269	NaN	NaN	
2020-03-05	Andorra	1.0	12.928	NaN	NaN	
...	...	...	...	...	...	
2021-11-05	Wallis and Futuna	NaN	NaN	NaN	NaN	
2021-11-05	World	249581640.0	31693.045	5039421.0	NaN	3.949
2021-11-05	Yemen	9843.0	322.820	1905.0	NaN	
2021-11-05	Zambia	209852.0	11091.158	3662.0	NaN	
2021-11-05	Zimbabwe	133112.0	8819.937	4685.0	NaN	3.356

Country selection

The countries are selected by the user, in this example we select the data of Belgium, France and Italy.

In [22]:

```
ArrOfLocations = ['Belgium','France','Italy']
Data = Data[Data['location'].isin(ArrOfLocations)]
```

In [23]:

```
Data:
```

Out[23]:

date	location	total_cases	total_cases_per_million	total_deaths	hosp_patients	people_vaccin
2020-03-05	Belgium	50.0	4.298	NaN	NaN	
2020-03-05	France	426.0	6.305	7.0	NaN	
2020-03-05	Italy	3858.0	63.909	148.0	2141.0	
2020-04-10	Belgium	26667.0	2292.489	3019.0	5663.0	
2020-04-10	France	56600.0	837.721	13199.0	31108.0	
...	...	...	...	...	...	
2021-10-25	France	7228331.0	106984.550	118452.0	6405.0	512247
2021-10-25	Italy	4743720.0	78580.731	131856.0	2917.0	465119
2021-11-05	Belgium	1403548.0	120659.190	26105.0	1931.0	87499
2021-11-05	France	7301303.0	108064.589	118830.0	6735.0	514598
2021-11-05	Italy	4795465.0	79437.898	132334.0	3519.0	467143

63 rows × 6 colums

Arranging the data

We also have to arrange the data in order to have a proper plot.

In [53]:

```
# arranging the data and renaming the columns
TotalCasesBelgium = Data[Data['location']=='Belgium'][['total_cases']].rename
(columns = {'total_cases':'Belgium'})
TotalCasesFrance = Data[Data['location']=='France'][['total_cases']].rename
(columns = {'total_cases':'France'})
TotalCasesItaly = Data[Data['location']=='Italy'][['total_cases']].rename
(columns = {'total_cases':'Italy'})
# merging the dataframes
TotalCases = TotalCasesBelgium.merge(TotalCasesFrance, on = 'date',
                                     how = "left").merge(TotalCasesItaly,
                                                         on = 'date', how = "left")
# filling missing values
TotalCases = TotalCases.fillna(method = "ffill")
```

In [54]:

```
TotalCases
```

Out[54]:

date	Belgium	France	Italy
2020-03-05	50.0	426.0	3858.0
2020-04-10	26667.0	56600.0	147577.0
2020-05-15	54644.0	181148.0	223885.0
2020-06-20	60550.0	200147.0	238275.0
2020-07-25	65727.0	220016.0	245864.0
2020-08-05	71158.0	233344.0	248803.0
2020-09-10	90568.0	394308.0	283180.0
2020-10-15	191959.0	852938.0	381602.0
2020-11-20	553680.0	2161970.0	1345767.0
2020-12-25	637246.0	2604584.0	2028354.0
2021-01-05	652735.0	2737892.0	2181619.0
2021-02-10	730951.0	3444827.0	2668266.0
2021-03-15	809861.0	4138369.0	3238394.0
2021-04-20	955056.0	5401341.0	3891063.0
2021-05-25	1050677.0	5670486.0	4197892.0
2021-06-05	1069874.0	5769287.0	4230153.0
2021-07-10	1093700.0	5870560.0	4269885.0
2021-08-15	1149869.0	6543481.0	4440669.0
2021-09-20	1224885.0	7045422.0	4638516.0

3.2.4 Plotting the data

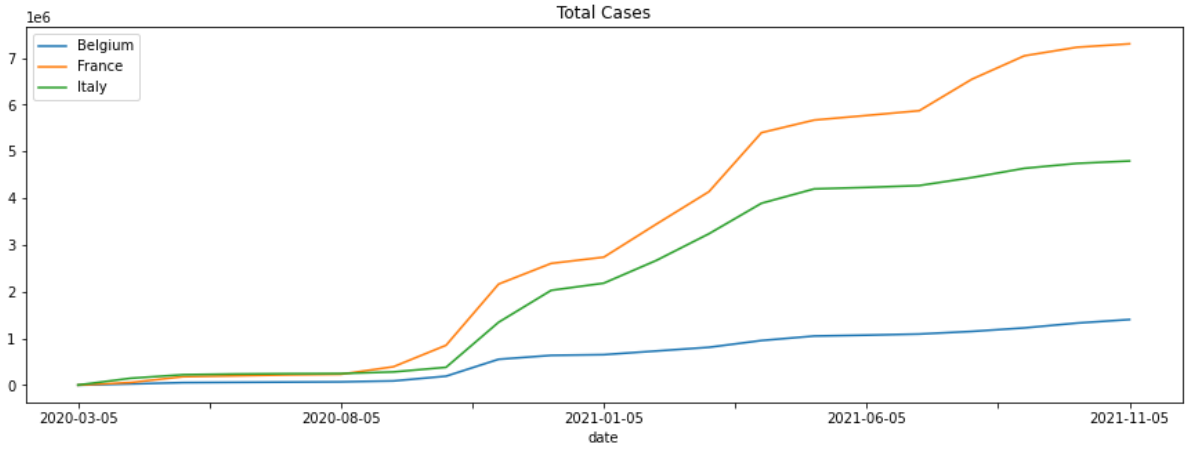
Line plot

In [66]:

```
TotalCases.plot(title = "Total Cases",figsize = (15,5))
```

Out[66]:

```
<AxesSubplot:title={'center':'Total Cases'}, xlabel='date'>
```



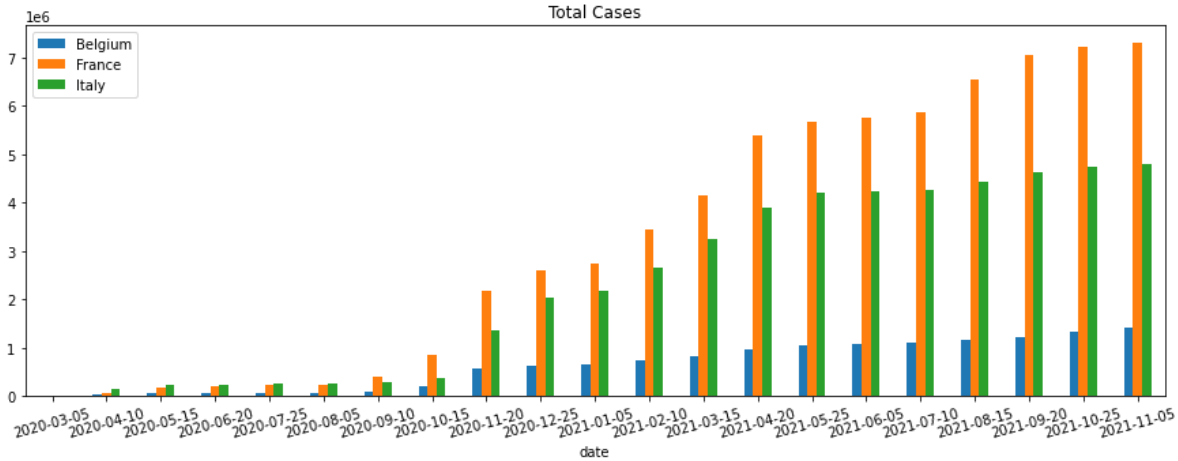
Bar plot

In [3]:

```
TotalCases.plot(kind = "bar",title = "Total Cases",figsize= (15,5),rot =15)
```

Out[3]:

```
<AxesSubplot:title={'center':'Total Cases'}, xlabel='date'>
```



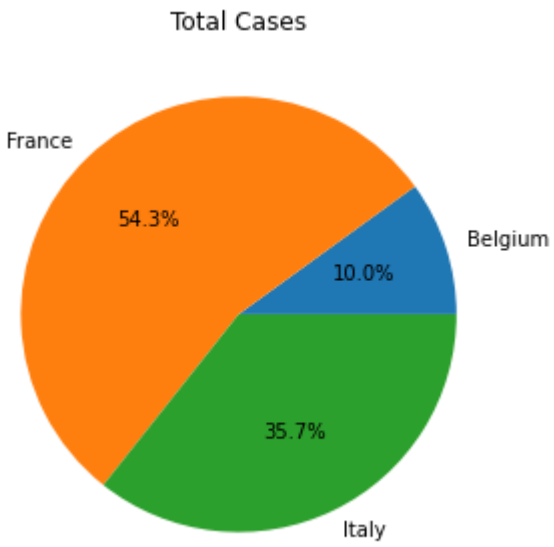
Pie plot

In [80]:

```
Numbers = TotalCases.iloc[19]
Numbers.plot(kind = "pie",title = "Total Cases",figsize= (15,5),
             autopct=lambda p: '{:.1f}%'.format(p) if p > 0 else '').
yaxis.set_label_text("")
```

Out[80]:

```
Text(0, 0.5, '')
```



4.0 Conclusion

In this project we managed two small application:

The first one is the image manipulation tool that can manipulate images chosen by the user. Thanks to this part we realized how many things we can do on images by simply converting them to a multidimensional numpy array. With this application the user will be able to choose the image from his disk, modify the chosen image and at the end save the modified image.

The second one is the covid data visualization tool, this application lets us visualize raw data that sometimes can be very hard to interpret. Data visualization can be very helpful for exploring data structure, detecting outliers and unusual groups, identifying trends and clusters, and much more. By representing the data with graphs we can read them more easily, the data becomes visual and understandable.

The given information for this project was clear and very helpful. More information about how to use properly the GUI will make this assignment easier to understand.