CSE_276C_HW4_P1

November 21, 2021

0.1 Homework 4 - CSE 276C - Math for Robotics

0.1.1 Problem 1

In robotics it is typical to have to recognize objects in the environment. We will here use the German Traffic Sign dataset for recognition of traffic signs. You can download the dataset from the link below.

To reduce computational time, please use the file Train subset.csv to read in the train set. Similarly, please use the file Test subset.csv to read in the test set.

Link: https://www.kaggle.com/meowmeowmeowmeowmeow/gtsrb-german-traffic-sign?select=Test.csv

Compute subspaces for the PCA and LDA methods. Provide illustration of the respective 1st and 2nd eigenvectors.

Compute the recognition rates for the test set. Report: - Correct classification - Incorrect classification

Provide at least one suggestion for how you might improve performance of each method.

0.1.2 Solution:

0.1.3 i.) Principal Component Analysis (PCA) Method with Random Forest Classifier for Image Recognition

1.) Import all the libraries that will be used.

[1]:
```python
# Import necessary libraries
import cv2
import math
import matplotlib
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns
from sklearn.decomposition import PCA
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.metrics import accuracy_score
```

2.) Load and store the "Train_subset.csv" and "Test_subset.csv" data as a dataframe.

[2]:
```python
# Read the subset of the train data.
train_subset_df = pd.read_csv('Train_subset.csv')

# Read the subset of the test data.
test_subset_df = pd.read_csv('Test_subset.csv')
```

3.) Observe how the “Test” dataframe and “Train” dataframe look like, and check whether the data are stored correctly. Print data information/statistics.

[3]:
```python
# First few number of row to print.
num_Row = 6

# Print the train dataframe to observe how they look like.
train_subset_df.head(num_Row)
```

[3]:
```
   Unnamed: 0  Width  Height  Roi.X1  Roi.Y1  Roi.X2  Roi.Y2  ClassId  \
0       19138     92      93       8       8      84      85       12
1       21703     70      61       7       6      64      56       13
2       32087     62      58       5       6      57      53       31
3       19762     47      49       6       6      42      44       12
4       13970     57      56       6       5      51      51        9
5       32376     32      31       5       6      27      26       31

                             Path
0  Train/12/00012_00025_00028.png
1  Train/13/00013_00041_00013.png
2  Train/31/00031_00000_00017.png
3  Train/12/00012_00046_00022.png
4   Train/9/00009_00013_00020.png
5  Train/31/00031_00010_00006.png
```

[4]:
```python
# Print the train dataframe statistics.
train_subset_df.describe()
```

[4]:
```
         Unnamed: 0         Width        Height        Roi.X1        Roi.Y1  \
count  10000.000000  10000.000000  10000.000000  10000.000000  10000.000000
mean   19661.150900     50.530100     49.988800      5.971900      5.941700
std    11317.849831     23.790807     22.627967      1.438651      1.351623
min        4.000000     25.000000     25.000000      4.000000      5.000000
25%     9894.750000     34.000000     34.000000      5.000000      5.000000
50%    19762.500000     43.000000     43.000000      6.000000      6.000000
75%    29507.500000     58.000000     57.000000      6.000000      6.000000
max    39208.000000    230.000000    203.000000     20.000000     18.000000

            Roi.X2       Roi.Y2       ClassId
count  10000.00000  10000.00000  10000.000000
mean      44.91260     44.41200     15.852400
std       22.59473     21.52965     12.002751
```

```
min       20.00000     20.00000     0.000000
25%       29.00000     29.00000     5.000000
50%       38.00000     38.00000    12.000000
75%       53.00000     52.00000    25.000000
max      211.00000    186.00000    42.000000
```

[5]:
```python
# Print the test dataframe to observe how they look like.
test_subset_df.head(num_Row)
```

[5]:
```
   Unnamed: 0  Width  Height  Roi.X1  Roi.Y1  Roi.X2  Roi.Y2  ClassId  \
0        6659     53      49       5       6      48      44       25
1        7633     44      42       5       6      39      37       11
2        1678     34      36       6       6      28      30       38
3        5938     38      42       5       6      33      37       38
4       11949     49      50       5       6      44      45        8
5        9991     34      36       6       5      29      30       38

             Path
0  Test/06659.png
1  Test/07633.png
2  Test/01678.png
3  Test/05938.png
4  Test/11949.png
5  Test/09991.png
```

[6]:
```python
# Print the train dataframe statistics.
test_subset_df.describe()
```

[6]:
```
         Unnamed: 0        Width       Height       Roi.X1       Roi.Y1  \
count   5000.000000  5000.000000  5000.00000  5000.000000  5000.000000
mean    6340.731200    50.371000    50.21100     5.983000     5.963200
std     3631.595137    25.191298    23.72901     1.557433     1.443559
min        5.000000    25.000000    25.00000     1.000000     5.000000
25%     3169.000000    34.000000    35.00000     5.000000     5.000000
50%     6435.000000    43.000000    43.00000     6.000000     6.000000
75%     9463.250000    58.000000    57.00000     6.000000     6.000000
max    12629.000000   260.000000   229.00000    22.000000    19.000000

            Roi.X2       Roi.Y2      ClassId
count  5000.000000  5000.000000  5000.000000
mean     44.736000    44.610400    15.489600
std      23.871512    22.518899    11.983347
min      20.000000    20.000000     0.000000
25%      29.000000    29.000000     5.000000
50%      38.000000    38.000000    12.000000
75%      52.000000    52.000000    25.000000
max     244.000000   210.000000    42.000000
```

4.) Store training and testing images and Class ID into respective lists.

[7]:
```
# List that stores train images
train_imgs = []
# List that store train class ID
train_IDs = []
# List that stores test images
test_imgs = []
# List that store test class ID
test_IDs = []

# Size of the image,
img_size = 25

# Store train images into the train images list
for idx in range(0, len(train_subset_df)):

    # Load image
    img = cv2.imread(train_subset_df['Path'][idx], cv2.IMREAD_COLOR)
    # Resize the image to 25 x 25 because the minimum width of the images are␣
 ↪25 x 25
    img = cv2.resize(img, (img_size, img_size))
    # Convert the image to grayscale
    img = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
    # Add to train images list
    train_imgs.append(img)

    # Get the ClassID
    train_class_ID = train_subset_df['ClassId'][idx]
    # Add to train ID list
    train_IDs.append(train_class_ID)

# Store test images into the test images list
for idx in range(0, len(test_subset_df)):

    # Load image
    img = cv2.imread(test_subset_df['Path'][idx], cv2.IMREAD_COLOR)
    # Resize the image to 25 x 25 because the minimum width of the images are␣
 ↪25 x 25
    img = cv2.resize(img, (img_size, img_size))
    # Convert the image to grayscale
    img = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
    # Add to train images list
    test_imgs.append(img)

    # Get the ClassID
    test_class_ID = test_subset_df['ClassId'][idx]
```

```
    # Add to train ID list
    test_IDs.append(test_class_ID)

## Convert the lists to arrays
# Train images and IDs arrays
train_imgs = np.array(train_imgs)
train_IDs = np.array(train_IDs)
#train_IDs = train_IDs.reshape(train_IDs.shape[0], 1)
# Test images and IDs arrays
test_imgs = np.array(test_imgs)
test_IDs = np.array(test_IDs)
#test_IDs = test_IDs.reshape(test_IDs.shape[0], 1)

# Print to check the shape of the arrays
print("Train Image Array Shape: ", train_imgs.shape)
print("Train IDs Array Shape: ", train_IDs.shape)
print("Test Image Array Shape: ", test_imgs.shape)
print("Train IDs Array Shape: ", test_IDs.shape)

print("\n----------------------------------------------\n")

# Print the total number of different traffic signs
class_IDs = np.unique(train_IDs)
nClasses = len(class_IDs)
print("Total Number of Different Traffic Signs: ", nClasses)
print("Traffic Sign IDs: ", class_IDs)
```

```
Train Image Array Shape:  (10000, 25, 25)
Train IDs Array Shape:  (10000,)
Test Image Array Shape:  (5000, 25, 25)
Train IDs Array Shape:  (5000,)

----------------------------------------------

Total Number of Different Traffic Signs:  43
Traffic Sign IDs:  [ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19
20 21 22 23
 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42]
```

5.) Scale or normalize the train and test images pixel values in between 0 and 1.

[8]:
```
# Normalize the train and test images pixel values.
train_imgs = train_imgs / 255.0
test_imgs = test_imgs / 255.0
```

6.) Reshape the train images dimensions from (Number of Images, N_{imgs} × Height of the Image, H × Width of the Image, W) to (N_{imgs} × HW).

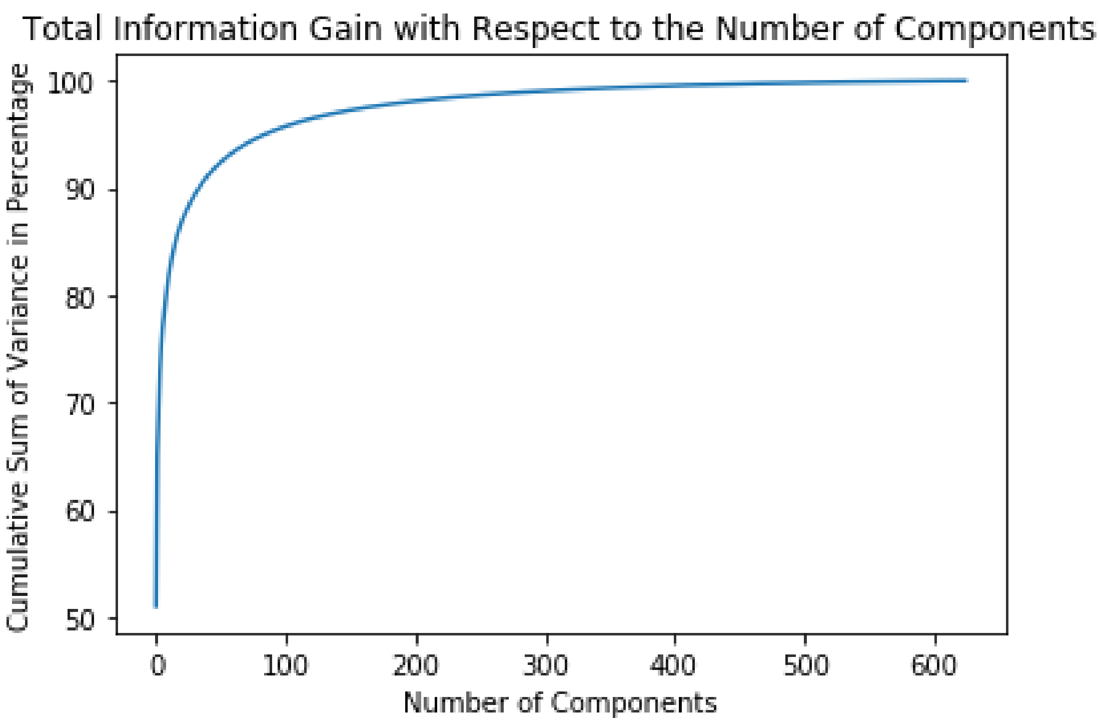
```
[9]: # Flatten the train images.
train_imgs_flatten = train_imgs.flatten().reshape(train_imgs.shape[0],␣
 ↪img_size*img_size)

# Flatten the test images.
test_imgs_flatten = test_imgs.flatten().reshape(test_imgs.shape[0],␣
 ↪img_size*img_size)

print("Flatten Train Images Shape: ", train_imgs_flatten.shape)
print("Flatten Test Images Shape: ", test_imgs_flatten.shape)
```

```
Flatten Train Images Shape:  (10000, 625)
Flatten Test Images Shape:  (5000, 625)
```

7.) Perform PCA method with the sklearn library to compress data.

```
[10]: # Determine the maximum number of components
n_Samples = train_imgs_flatten.shape[0] # Number of Samples
n_Features = train_imgs_flatten.shape[1] # Number of Features

# If the number of samples is more than the number of Features.
if n_Samples <= n_Features:

    # Maximum of the number of components is the number of samples - 1.
    max_n_components = n_Samples - 1

else:

    # Maximum of the number of components is the total number of features.
    max_n_components = img_size*img_size

## Plot the total information gains with respect to the number of components.
# Create the PCA object.
pca = PCA(n_components = max_n_components)
# Fit the flatten train images array
pca.fit(train_imgs_flatten)
# Plot the graph
plt.plot(np.cumsum(pca.explained_variance_ratio_*100))
# Include title of the graph
plt.title("Total Information Gain with Respect to the Number of Components")
# Label x-axis
plt.xlabel("Number of Components")
# Label y-axis
plt.ylabel("Cumulative Sum of Variance in Percentage")
# Show plot
plt.show()
```

```python
# Select the number of components that contains 96% of the variance or 0.96␣
↪variance
variance_num = 0.96
# Create the PCA object.
pca = PCA(variance_num)
# Fit the flatten train images array
pca.fit(train_imgs_flatten)
# Print the total number of components need to get 96% of the variance
num_PCA_components = pca.n_components_
print("Number of Components Needed to Capture 96% of the Information: ",␣
↪num_PCA_components)

# Create the PCA object
pca = PCA(n_components = num_PCA_components)
# Fit and transform train images
x_train = pca.fit_transform(train_imgs_flatten)
# Transform test images
x_test = pca.transform(test_imgs_flatten)
```




```
Number of Components Needed to Capture 96% of the Information:  108
```

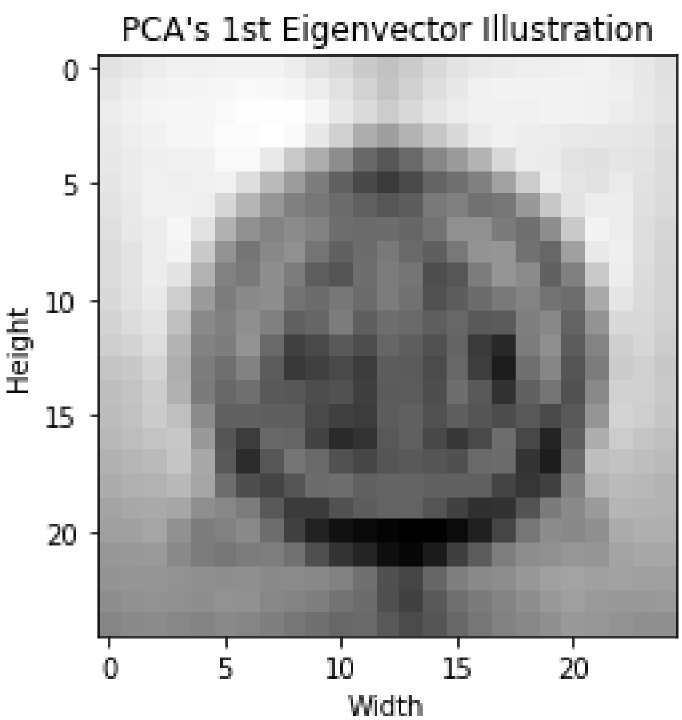
8.) Provide illustration of the respective 1st and 2nd eigenvectors.

[11]:
```python
# Get the first and second eigenvectors
first_eigenVector = pca.components_[0]
second_eigenVector = pca.components_[1]

# Illustrate the 1st Eigenvector of PCA
print("PCA's 1st Eigenvector:\n", first_eigenVector)
plt.title("PCA's 1st Eigenvector Illustration")
plt.xlabel("Width")
plt.ylabel("Height")
plt.imshow(first_eigenVector.reshape((img_size, img_size)), cmap ='gray')
plt.show()
```

```
PCA's 1st Eigenvector:
 [0.04672869 0.04722522 0.0477394  0.04814982 0.0480401  0.04833528
 0.04843048 0.04838607 0.04776281 0.04662202 0.0457755  0.04434811
 0.04341673 0.04440314 0.04582263 0.04689159 0.04734314 0.04761885
 0.0479059  0.04792918 0.0480634  0.04825382 0.04782014 0.04725342
 0.04651578 0.04719962 0.04780766 0.04821571 0.04860554 0.04852101
 0.04878519 0.04899623 0.0488187  0.04857497 0.04770112 0.04660279
 0.04510817 0.04374004 0.04490786 0.04631801 0.04727816 0.04808413
 0.04842625 0.04822143 0.04834036 0.0484686  0.04817547 0.04798324
 0.04729098 0.04668859 0.04764603 0.04808231 0.04852513 0.04878684
 0.04868844 0.04894196 0.04934266 0.04936477 0.04936685 0.04867622
 0.04732229 0.04589013 0.0445565  0.04566276 0.04697191 0.04765082
 0.0481743  0.04815966 0.04806122 0.04833981 0.04835577 0.04821319
 0.04772706 0.0473538  0.04686403 0.04737608 0.04791117 0.0481224
 0.04862475 0.04873782 0.04917829 0.04918229 0.04958764 0.04901582
 0.04732689 0.0449268  0.04144551 0.0400576  0.0420993  0.04406134
 0.04580912 0.04755144 0.04814902 0.0478151  0.04764278 0.04737274
 0.04730066 0.04714394 0.04698474 0.04633694 0.04723581 0.04759812
 0.04799969 0.04805051 0.04819897 0.04895005 0.04890185 0.04736461
 0.04423742 0.04151113 0.03837254 0.03467923 0.03296193 0.03481769
 0.03776178 0.03969341 0.04222228 0.04567614 0.04764026 0.04758579
 0.04678571 0.04647745 0.04694263 0.04680147 0.04610277 0.04706082
 0.04758988 0.04810581 0.04791638 0.04803004 0.04830716 0.04631989
 0.04280444 0.03984178 0.03673593 0.03415447 0.03162167 0.03017206
 0.03180406 0.0342609  0.0354666  0.03704558 0.04035184 0.04449149
 0.04779192 0.04699537 0.0465728  0.04748488 0.04661645 0.04601225
 0.04676913 0.04741392 0.0480393  0.04819548 0.04782175 0.04563631
 0.04222783 0.03927903 0.03711226 0.03618814 0.03505901 0.03388173
 0.03293453 0.03366935 0.03640945 0.03708258 0.03574353 0.03609681
 0.03948608 0.04408897 0.0469198  0.04774062 0.04774379 0.0464988
 0.04591104 0.04639869 0.04725993 0.04769145 0.04862483 0.04667912
 0.04192805 0.03861903 0.03733203 0.03816463 0.03729388 0.03528469
 0.03499184 0.03514306 0.03453172 0.03583607 0.03862089 0.03868074
 0.03654607 0.03543755 0.03848636 0.04427571 0.04847885 0.04763989
 0.04619922 0.04551612 0.04614875 0.0468018  0.04767173 0.04836256
 0.04432565 0.03903057 0.03601138 0.03788626 0.03865628 0.03584496
```

0.03416605	0.03511446	0.03648678	0.03478622	0.03383485	0.03633311
0.03893026	0.03916555	0.03574576	0.03465413	0.04019653	0.04733793
0.04791916	0.04629512	0.04537652	0.04609762	0.04691598	0.04775129
0.0468033	0.04201519	0.03713437	0.03652501	0.03846688	0.03664554
0.03372007	0.03275737	0.03530581	0.03666284	0.0359833	0.03217928
0.03295179	0.03675885	0.03911011	0.03811555	0.03399891	0.037092
0.04424234	0.04778768	0.04602993	0.04522854	0.04579022	0.04655093
0.04742109	0.04470694	0.03976086	0.03667376	0.03801274	0.03845095
0.03575614	0.03514286	0.03621799	0.03496437	0.03663152	0.03565719
0.03249095	0.03362685	0.03512866	0.0364149	0.03749009	0.0357745
0.03516762	0.04108605	0.04698459	0.04590794	0.04510152	0.0452429
0.04606426	0.04692643	0.0433197	0.03800294	0.03706933	0.03850692
0.03708994	0.03393035	0.03453771	0.03676901	0.03381796	0.03529711
0.03477596	0.03369583	0.03456934	0.03331114	0.0334697	0.03699439
0.03779347	0.03424502	0.03890901	0.0458462	0.0457317	0.04491117
0.04473891	0.04528191	0.04617231	0.04206028	0.03722595	0.03678487
0.03902731	0.03484059	0.03095207	0.03170274	0.03320355	0.03206385
0.03447062	0.0337972	0.03257195	0.03431813	0.03018577	0.02847318
0.03666741	0.03853592	0.03354223	0.03751099	0.04483171	0.04546805
0.04452662	0.0438611	0.04410496	0.0452188	0.04158518	0.03677854
0.03544785	0.0372834	0.03352382	0.03008332	0.03075166	0.03168845
0.03042659	0.03348836	0.03345297	0.03191297	0.0347117	0.03061796
0.02723408	0.03544802	0.0376137	0.03279683	0.03667322	0.04446443
0.04493528	0.04418165	0.04313456	0.04367626	0.04458873	0.0419029
0.03658347	0.03470642	0.03544235	0.03334659	0.0330677	0.03211284
0.03253834	0.03066199	0.03342644	0.03354211	0.03219124	0.03529766
0.03331625	0.02943652	0.03413697	0.0360161	0.03250853	0.037951
0.04500937	0.0448822	0.04421175	0.04300521	0.0434474	0.04446412
0.04330181	0.03813013	0.03405476	0.0340511	0.03383963	0.03379542
0.03171394	0.03075126	0.03045781	0.03355182	0.03416284	0.0324183
0.03379356	0.03280197	0.03198915	0.03310457	0.03169447	0.0331087
0.03917476	0.04525939	0.04471486	0.04390754	0.04255001	0.0430712
0.04370992	0.04407827	0.03955144	0.03287638	0.03046684	0.03468515
0.03443635	0.03104279	0.02859272	0.02952679	0.0331401	0.0339308
0.03155216	0.0299373	0.03175764	0.03532436	0.03150192	0.02805157
0.03380357	0.04150029	0.04457323	0.04383793	0.04322364	0.04190346
0.04233293	0.04281242	0.04398917	0.04097271	0.03330458	0.02885462
0.03294161	0.03588256	0.03488029	0.03245872	0.03135772	0.03287398
0.03328089	0.03281582	0.03236826	0.03485591	0.03509715	0.0296987
0.02731314	0.03508111	0.0431425	0.0435476	0.04318861	0.04287993
0.04142962	0.04176715	0.04193236	0.0426905	0.04072899	0.03518231
0.03215068	0.03119065	0.03271996	0.03514404	0.03526263	0.03407716
0.03432226	0.03436289	0.0341022	0.0341353	0.0339041	0.0311928
0.02984523	0.03100966	0.03705012	0.04201217	0.04287665	0.04256489
0.04209236	0.04060768	0.04099818	0.04123109	0.04079746	0.03855652
0.03777988	0.036644	0.03335794	0.03034306	0.03031296	0.03254588
0.03365108	0.0342616	0.03423116	0.03277875	0.03092596	0.02932408
0.02942385	0.03264369	0.03657945	0.03804381	0.03968805	0.04226871

```
0.04214472 0.04193323 0.04010109 0.04019629 0.04069534 0.03876672
0.03674803 0.03719753 0.03793667 0.03521123 0.03101037 0.02785131
0.02616029 0.02535117 0.02493422 0.02445688 0.02457731 0.02562326
0.02767578 0.03127244 0.03604284 0.03825412 0.03783388 0.03853249
0.0413291  0.04167517 0.04154002 0.03931297 0.03950796 0.03965995
0.03794061 0.03672199 0.03613793 0.03670178 0.0369151  0.03558983
0.03224838 0.02967879 0.02805561 0.02584474 0.02524168 0.02713932
0.03064985 0.03351792 0.03683607 0.03834772 0.03862854 0.0392426
0.03917511 0.04046619 0.0409752  0.04088428 0.03889689 0.03914635
0.03909966 0.03879581 0.03850932 0.03849349 0.03855464 0.03809776
0.03815977 0.03786672 0.03713337 0.03547212 0.03236845 0.03130301
0.03452643 0.03695672 0.03798498 0.03830757 0.03929617 0.04023163
0.0405395  0.04024125 0.04032224 0.04014618 0.04014456 0.03831795
0.0387254  0.03881015 0.03860453 0.03840436 0.03879667 0.03870058
0.03779652 0.0373807  0.03699729 0.03575187 0.03510926 0.03227549
0.03096299 0.03332915 0.03542697 0.0365419  0.03745109 0.03844723
0.03947973 0.04013123 0.03965834 0.0394256  0.03944982 0.03966959
0.03760711 0.03807693 0.03815637 0.03775396 0.03743312 0.03792512
0.03757102 0.03701196 0.03615726 0.03545495 0.03465338 0.0348659
0.03318741 0.03190895 0.03282235 0.03461628 0.03604368 0.03716405
0.03784547 0.03852682 0.03948814 0.03929838 0.03896048 0.03857894
0.0385147 ]
```

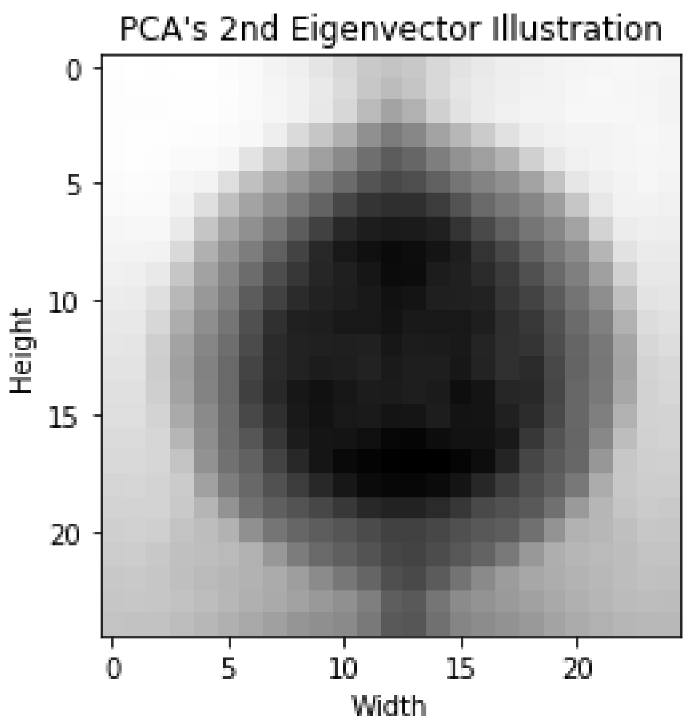


[12]:
```python
# Illustrate the 2nd Eigenvector of PCA
print("PCA's 2nd Eigenvector:\n", second_eigenVector)
plt.title("PCA's 2nd Eigenvector Illustration")
plt.xlabel("Width")
plt.ylabel("Height")
plt.imshow(second_eigenVector.reshape((img_size, img_size)), cmap ='gray')
plt.show()
```

```
PCA's 2nd Eigenvector:
 [ 0.04897201  0.04936645  0.04923463  0.04983891  0.04952082  0.0480383
  0.04640888  0.04417447  0.04178622  0.03680512  0.02995201  0.02260445
  0.01948271  0.02211666  0.03018846  0.03502242  0.03882105  0.04103913
  0.04219225  0.04336432  0.04447479  0.04533673  0.04565435  0.04510694
  0.04457517  0.04851385  0.04899113  0.04901221  0.04926246  0.04913951
  0.04845858  0.0464944   0.04480837  0.04247827  0.03885763  0.03155188
  0.02174544  0.01619792  0.02066156  0.03029374  0.03666696  0.03945994
  0.04137537  0.04322464  0.04398183  0.04504924  0.04602445  0.04530535
  0.04476355  0.04382245  0.04879333  0.0488048   0.04869017  0.04911163
  0.04928023  0.04803755  0.04681178  0.04508667  0.0425819   0.03828939
  0.02973498  0.01502237  0.00441508  0.01080391  0.02621     0.0353681
  0.03980659  0.04208305  0.0436245   0.04406601  0.04549113  0.04572239
  0.04571426  0.0452752   0.04433165  0.04866496  0.0488444   0.0490718
  0.04851808  0.048282    0.04720037  0.04602364  0.04108558  0.0311975
  0.02097468  0.01013343 -0.00588121 -0.01606676 -0.01041793  0.0038225
  0.01366898  0.02347752  0.03465252  0.040446    0.0427053   0.04360609
  0.04409639  0.04451714  0.04538301  0.04459266  0.04854221  0.0490086
  0.048542    0.04689627  0.0467486   0.04494299  0.03625899  0.0225898
  0.01144975  0.00184519 -0.00881658 -0.02217648 -0.03143784 -0.02663489
 -0.01455208 -0.00497134  0.00168877  0.01176468  0.0259097   0.03819272
  0.04204674  0.04357187  0.04443647  0.04534825  0.04438637  0.04738906
  0.04785826  0.04767537  0.04520613  0.0434331   0.03305118  0.01814929
  0.00553149 -0.00308589 -0.01304882 -0.02422458 -0.03576782 -0.04070104
 -0.03884122 -0.02948856 -0.01934791 -0.01210075 -0.00577999  0.0041017
  0.02038663  0.03713912  0.04178175  0.04387433  0.04487443  0.04360222
  0.04649327  0.04697985  0.04670745  0.04322156  0.03364159  0.01761152
  0.00386473 -0.00620983 -0.01680381 -0.02838348 -0.04261633 -0.05128746
 -0.05391646 -0.0539529  -0.04743847 -0.03591718 -0.02486523 -0.01723348
 -0.0106637   0.00122706  0.02135086  0.03742761  0.04290171  0.04327087
  0.04224239  0.04523301  0.04577936  0.0453678   0.03933312  0.02048885
  0.00458176 -0.00585809 -0.01805312 -0.03146873 -0.0450991  -0.057334
 -0.0628738  -0.06456098 -0.06432545 -0.06131525 -0.05206979 -0.04067114
 -0.02954413 -0.019433   -0.01017866  0.00411409  0.02751302  0.04030958
  0.04174467  0.04095407  0.04343784  0.04387623  0.04307359  0.03078289
  0.01034907 -0.00388928 -0.01534217 -0.02975351 -0.04402068 -0.0573804
 -0.06576006 -0.06942309 -0.07270793 -0.07101379 -0.06778714 -0.06269587
 -0.05204524 -0.04168729 -0.02884268 -0.01743678 -0.00800914  0.01293224
  0.03519579  0.03984467  0.03910199  0.04205674  0.04183721  0.03873605
  0.02097783  0.00344867 -0.01007814 -0.02372692 -0.03891051 -0.05018895
```

-0.05869087	-0.06197002	-0.06659078	-0.07224162	-0.07147174	-0.06369179
-0.06121204	-0.05650264	-0.04789266	-0.03763882	-0.02453888	-0.01434965
0.00150646	0.02733611	0.0376232	0.03786398	0.03994898	0.03969758
0.03356012	0.01406368	-0.00187137	-0.01441701	-0.03189271	-0.04513024
-0.05596575	-0.06077451	-0.06266553	-0.06527943	-0.06907202	-0.06778593
-0.06216912	-0.06141433	-0.06049075	-0.05083871	-0.04432448	-0.03222142
-0.01984849	-0.00732329	0.01837003	0.03547337	0.0366064	0.03822141
0.03823232	0.02967694	0.00852166	-0.00698183	-0.01913378	-0.0378367
-0.05318276	-0.06150251	-0.06250989	-0.06538643	-0.06582557	-0.06815599
-0.06521741	-0.06548034	-0.06534457	-0.06280528	-0.05416021	-0.04915095
-0.03672453	-0.02438992	-0.01360922	0.01059971	0.03238206	0.03468489
0.03729614	0.03670272	0.02576979	0.00379209	-0.01085885	-0.02263576
-0.04018553	-0.05611082	-0.06071848	-0.06104965	-0.06194691	-0.06169443
-0.06608131	-0.06339247	-0.06403719	-0.06523324	-0.05957832	-0.05303508
-0.05177827	-0.03927195	-0.02718798	-0.0176044	0.00496025	0.02963242
0.03251934	0.03614985	0.03538685	0.0246996	0.00222111	-0.01270968
-0.0251255	-0.0434132	-0.05651371	-0.06077149	-0.06337533	-0.06250469
-0.060632	-0.06451018	-0.06270983	-0.06273675	-0.06603834	-0.06007171
-0.05315099	-0.0564093	-0.04164266	-0.02755202	-0.01800962	0.0038751
0.02803548	0.03034871	0.03438127	0.0339148	0.02584855	0.00376809
-0.01156797	-0.02452828	-0.04544561	-0.05719526	-0.06616306	-0.0691605
-0.06605347	-0.06343853	-0.06399059	-0.06270962	-0.06414426	-0.07066089
-0.06734483	-0.05847519	-0.05771008	-0.04264954	-0.02629431	-0.01691696
0.00584142	0.02739097	0.02944869	0.03297906	0.03290946	0.02806827
0.00718789	-0.00877571	-0.023711	-0.04157464	-0.05387629	-0.0653083
-0.06864799	-0.06542917	-0.06447607	-0.06741493	-0.0672537	-0.06298043
-0.06826446	-0.067689	-0.06176892	-0.05496676	-0.04085551	-0.02606618
-0.01436606	0.01072945	0.02692608	0.02820781	0.03140924	0.03158122
0.02934783	0.0131035	-0.00664613	-0.02328887	-0.03490594	-0.04993987
-0.06338533	-0.06725545	-0.06753534	-0.06994902	-0.07467175	-0.07493022
-0.07088341	-0.06841	-0.06880633	-0.06486693	-0.05006825	-0.03637607
-0.02676596	-0.00996375	0.01658181	0.02648103	0.02750324	0.03055154
0.03082503	0.02932368	0.02020635	-0.00494614	-0.02087526	-0.02934445
-0.04407251	-0.0576131	-0.06714699	-0.07315781	-0.07544495	-0.07707343
-0.07779702	-0.07743544	-0.0752681	-0.07101516	-0.05958577	-0.04275389
-0.03253683	-0.0259532	-0.0022549	0.02167257	0.02581495	0.02600249
0.02887427	0.02917989	0.02813553	0.02375258	0.0028497	-0.01505034
-0.02569676	-0.03649354	-0.04694568	-0.05695782	-0.06594564	-0.07184953
-0.07406333	-0.07433693	-0.07194442	-0.06754131	-0.05821874	-0.04673373
-0.03758714	-0.03015649	-0.01794961	0.00824063	0.02241564	0.02487153
0.02444013	0.0276651	0.02802286	0.02748775	0.02244755	0.01277573
-0.00433145	-0.02084987	-0.03032796	-0.03610933	-0.04192214	-0.04927268
-0.05755409	-0.06057902	-0.06038681	-0.05672286	-0.04987773	-0.0412362
-0.03676252	-0.03307153	-0.0217812	-0.00094596	0.01344042	0.02063932
0.02344869	0.02278059	0.02587254	0.02681123	0.02581128	0.01994394
0.01684997	0.01107136	-0.00569741	-0.01927856	-0.02648532	-0.0310948
-0.03510572	-0.04039095	-0.04439985	-0.04460995	-0.04225594	-0.03766044
-0.03387039	-0.03109909	-0.02194663	-0.00367603	0.01069554	0.01361633

```
 0.01775325  0.02146913  0.02114356  0.02394399  0.02448065  0.02231871
 0.01804279  0.01703717  0.01541141  0.01079819 -0.00194903 -0.01583328
-0.02487217 -0.03031473 -0.03640768 -0.04261023 -0.04398373 -0.03977089
-0.03420643 -0.0282445  -0.01681015 -0.00167558  0.00934255  0.01319791
 0.01516746  0.01691709  0.02012854  0.02035799  0.02234343  0.02253792
 0.0209807   0.01752603  0.01524281  0.01325747  0.01107997  0.00789991
 0.00138383 -0.00814867 -0.01815724 -0.02699124 -0.03819918 -0.04090723
-0.03194301 -0.01996425 -0.00898363  0.00029352  0.00550107  0.00887079
 0.01186381  0.01427311  0.01659188  0.01880314  0.01797518  0.02118326
 0.02158796  0.0211731   0.01933285  0.01652353  0.01372552  0.01028504
 0.00712654  0.00307266 -0.00106526 -0.00577951 -0.01432888 -0.03087918
-0.03317808 -0.01906081 -0.00831242 -0.00406569 -0.00052742  0.0026104
 0.00654532  0.01019829  0.01343032  0.01550739  0.01650091  0.01615454
 0.01971115  0.01935479  0.01931974  0.016845    0.01347574  0.01020011
 0.00643054  0.00218455 -0.00315306 -0.0061644  -0.00868726 -0.0170167
-0.03246391 -0.03456701 -0.02152675 -0.01135236 -0.00766695 -0.00451035
-0.00108703  0.003526    0.00753746  0.01065099  0.01329378  0.01409663
 0.01409214]
```



PCA Eigenvectors Explanation: The PCA eigenvectors are the principal components from the training images, which are images of a traffic sign. Each traffic sign is also a weighted combination of the eigen images as illustrated above.

9.) Apply Random Forest Classification to perform image recognition.

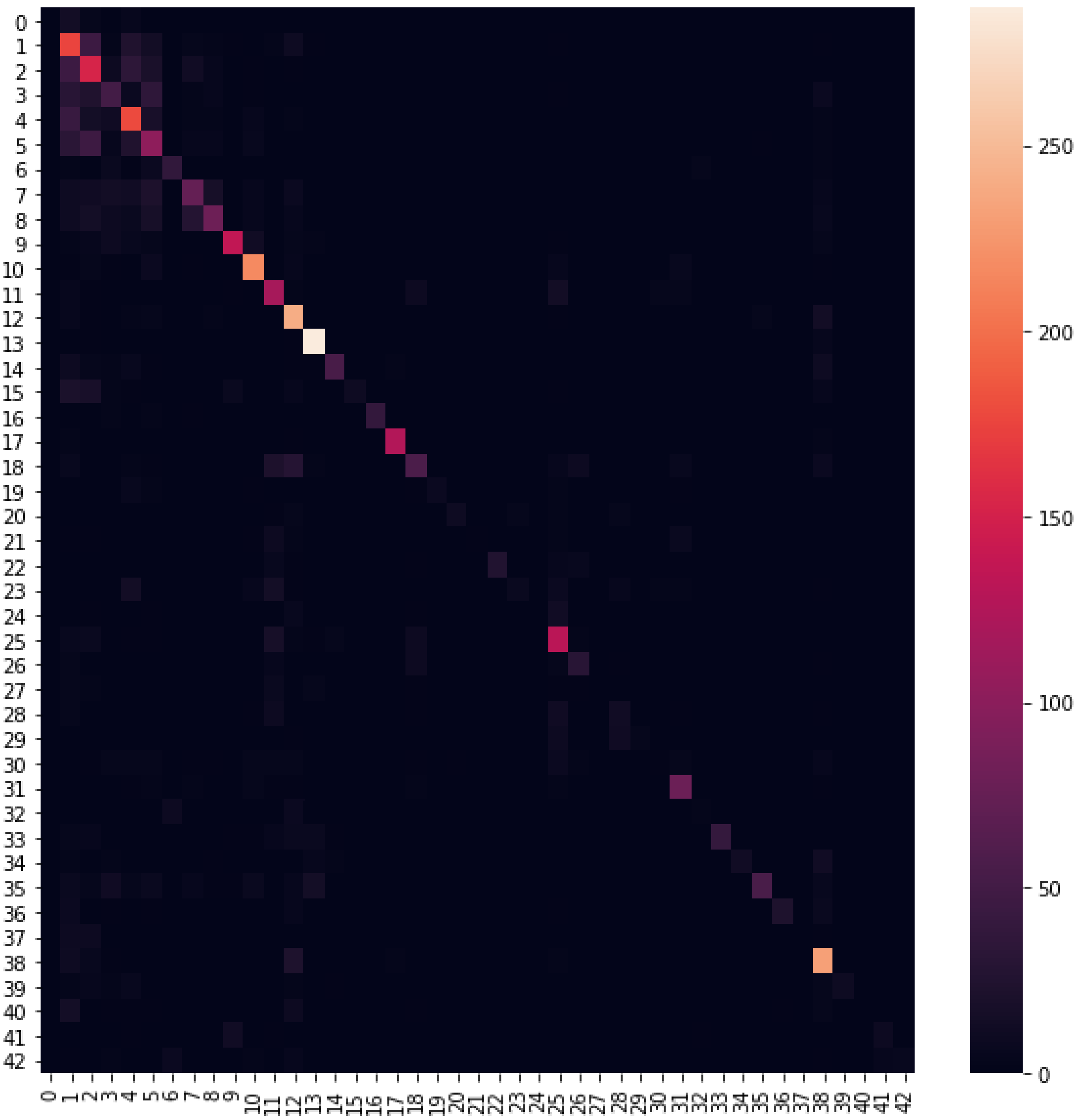
```
[13]: # Create Random Forest Classifier Object
RF_classifier = RandomForestClassifier()

# Fit the flatten train images with the train IDs
RF_classifier.fit(x_train, train_IDs)

# Predict the test set outcomes.
y_predict = RF_classifier.predict(x_test)
```

10.) Compute the PCA recognition rates for the test set to report correct classification and incorrect classification.

```
[14]: # Plot Confusion Matrix
Confusion_Matrix = confusion_matrix(test_IDs, y_predict)
print("Confusion Matrix: \n\n", Confusion_Matrix)

# Show Heatmap
plt.figure(figsize=(10,10))
sns.heatmap(Confusion_Matrix)

# Print Classification Report
print("\n\nClassfication Report:␣
 ↪\n\n",classification_report(test_IDs,y_predict))
```

```
Confusion Matrix:

 [[  0  14   4 …   0   0   0]
 [  0 176  45 …   0   0   0]
 [  0  45 153 …   0   0   0]
 …
 [  0  15   1 …   1   0   0]
 [  0   0   0 …   0   9   0]
 [  0   2   0 …   0   5   6]]


Classfication Report:

               precision    recall  f1-score   support

           0       0.00      0.00      0.00        25
           1       0.35      0.60      0.44       291
           2       0.38      0.53      0.44       287
           3       0.31      0.29      0.30       174
           4       0.49      0.63      0.55       285
           5       0.35      0.44      0.39       234
           6       0.69      0.56      0.62        66
           7       0.50      0.41      0.45       185
           8       0.58      0.44      0.50       183
```

```
           9       0.80      0.71      0.75       191
          10       0.74      0.84      0.79       257
          11       0.50      0.72      0.59       165
          12       0.57      0.86      0.69       281
          13       0.85      0.96      0.90       299
          14       0.82      0.54      0.65       100
          15       0.79      0.14      0.23        80
          16       0.97      0.80      0.88        49
          17       0.90      0.93      0.91       136
          18       0.53      0.37      0.44       155
          19       0.80      0.29      0.42        28
          20       0.73      0.31      0.44        35
          21       1.00      0.06      0.11        33
          22       1.00      0.50      0.67        50
          23       0.54      0.10      0.17        68
          24       0.00      0.00      0.00        33
          25       0.54      0.68      0.60       194
          26       0.54      0.51      0.53        61
          27       0.00      0.00      0.00        25
          28       0.33      0.25      0.29        51
          29       1.00      0.12      0.22        33
          30       0.20      0.03      0.06        58
          31       0.68      0.75      0.71       105
          32       0.33      0.15      0.21        20
          33       0.98      0.51      0.67        79
          34       0.93      0.28      0.43        47
          35       0.86      0.42      0.57       132
          36       0.92      0.45      0.61        51
          37       0.00      0.00      0.00        26
          38       0.62      0.81      0.70       284
          39       1.00      0.28      0.44        39
          40       1.00      0.02      0.05        41
          41       0.64      0.31      0.42        29
          42       1.00      0.17      0.29        35

    accuracy                           0.57      5000
   macro avg       0.62      0.41      0.44      5000
weighted avg       0.60      0.57      0.55      5000

/Users/kaitheuser/opt/anaconda3/lib/python3.7/site-
packages/sklearn/metrics/_classification.py:1272: UndefinedMetricWarning:
Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

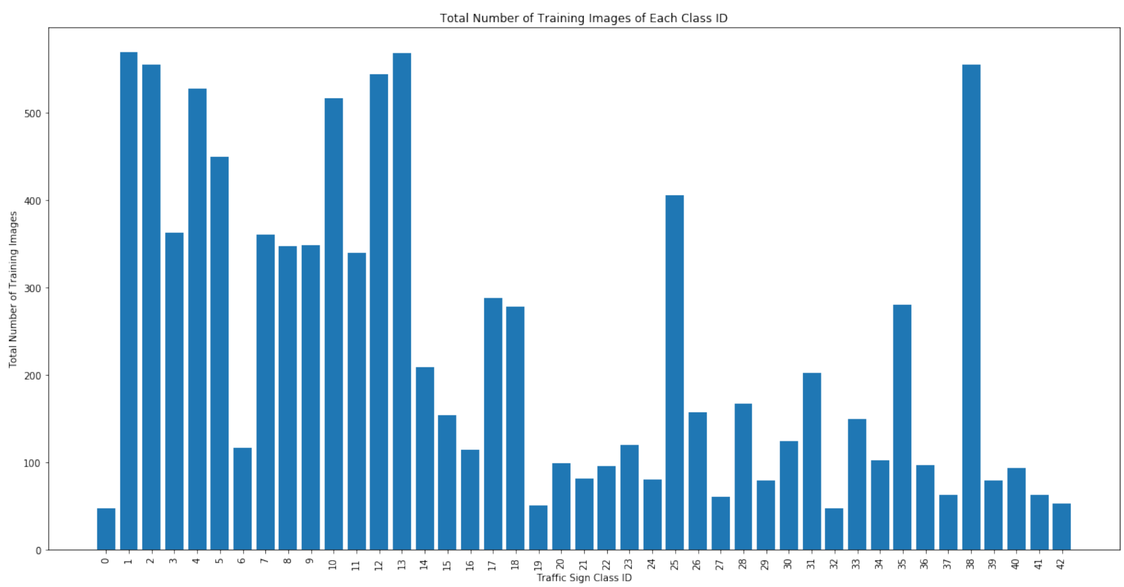


[15]:
```
# Total Number of Test Images
total_num_test_imgs = np.sum(Confusion_Matrix)
# Total Number of Correct Classification
tot_num_correct_classification = np.trace(Confusion_Matrix)
# Total Number of Incorrect Classification
tot_num_incorrect_classification = total_num_test_imgs - tot_num_correct_classification

# Print Recognition Rate Report
print('Total Number of Test Images: ', total_num_test_imgs)
print("Total Number of Correct Classifications: ", tot_num_correct_classification)
```

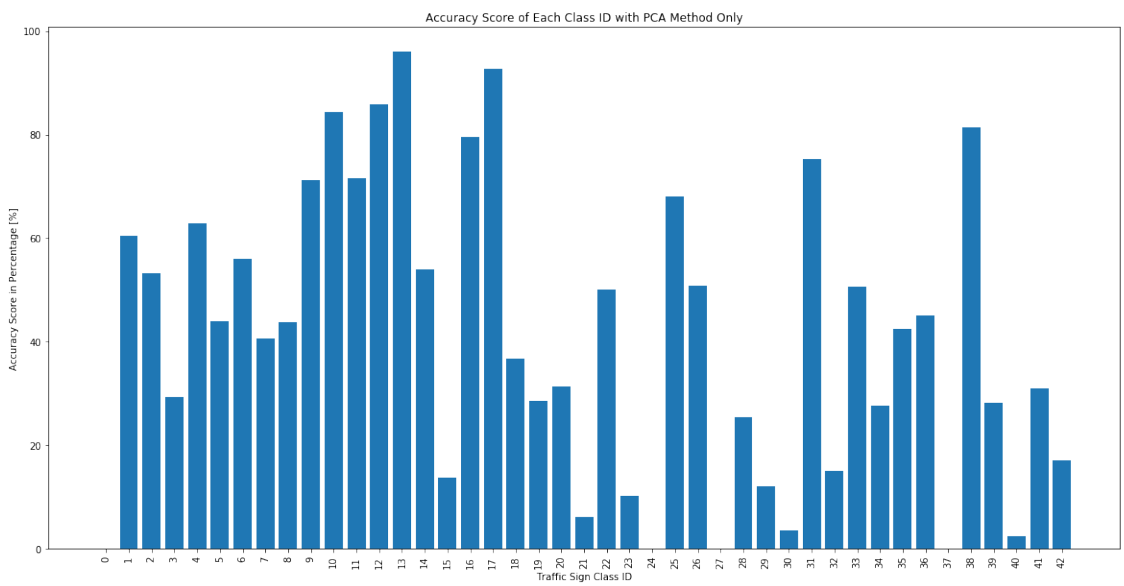
```
print("Total Number of Incorrect Classifications: ",␣
↪tot_num_incorrect_classification)
print('\nAccuracy Score: ' + str(accuracy_score(test_IDs, y_predict)*100) + '␣
↪%')
```

```
Total Number of Test Images:  5000
Total Number of Correct Classifications:  2847
Total Number of Incorrect Classifications:  2153

Accuracy Score: 56.940000000000005 %
```

[16]:
```
class_IDs_freq = np.zeros((1, nClasses))

# Store train images into the train images list
for idx in range(0, len(train_subset_df)):

    # Get the ClassID
    train_class_ID = train_subset_df['ClassId'][idx]
    # Count the numbers of different Class ID
    class_IDs_freq[0, train_class_ID] = class_IDs_freq[0, train_class_ID] + 1

plt.figure(figsize=(20,10))
plt.title('Total Number of Training Images of Each Class ID')
plt.ylabel('Total Number of Training Images')
plt.xlabel('Traffic Sign Class ID')
plt.bar(class_IDs, class_IDs_freq[0])
plt.xticks(class_IDs, rotation='vertical')
plt.show()
```



[17]:
```
class_IDs_accuracy_score = np.zeros((1, nClasses))

# Calculate the correct classification percentage of each class
for idx in range(0, Confusion_Matrix.shape[0]):

    # Get the the sum of Class IDs
    sum_of_IDs = np.sum(Confusion_Matrix[idx,:])

    # Get the accuracy score in percentage
    class_IDs_accuracy_score[0, idx] = Confusion_Matrix[idx,idx] / sum_of_IDs * 100

plt.figure(figsize=(20,10))
plt.title('Accuracy Score of Each Class ID with PCA Method Only')
plt.ylabel('Accuracy Score in Percentage [%]')
plt.xlabel('Traffic Sign Class ID')
plt.bar(class_IDs, class_IDs_accuracy_score[0])
plt.xticks(class_IDs, rotation='vertical')
plt.show()
```



Suggestion to Improve the PCA Recognition Rate

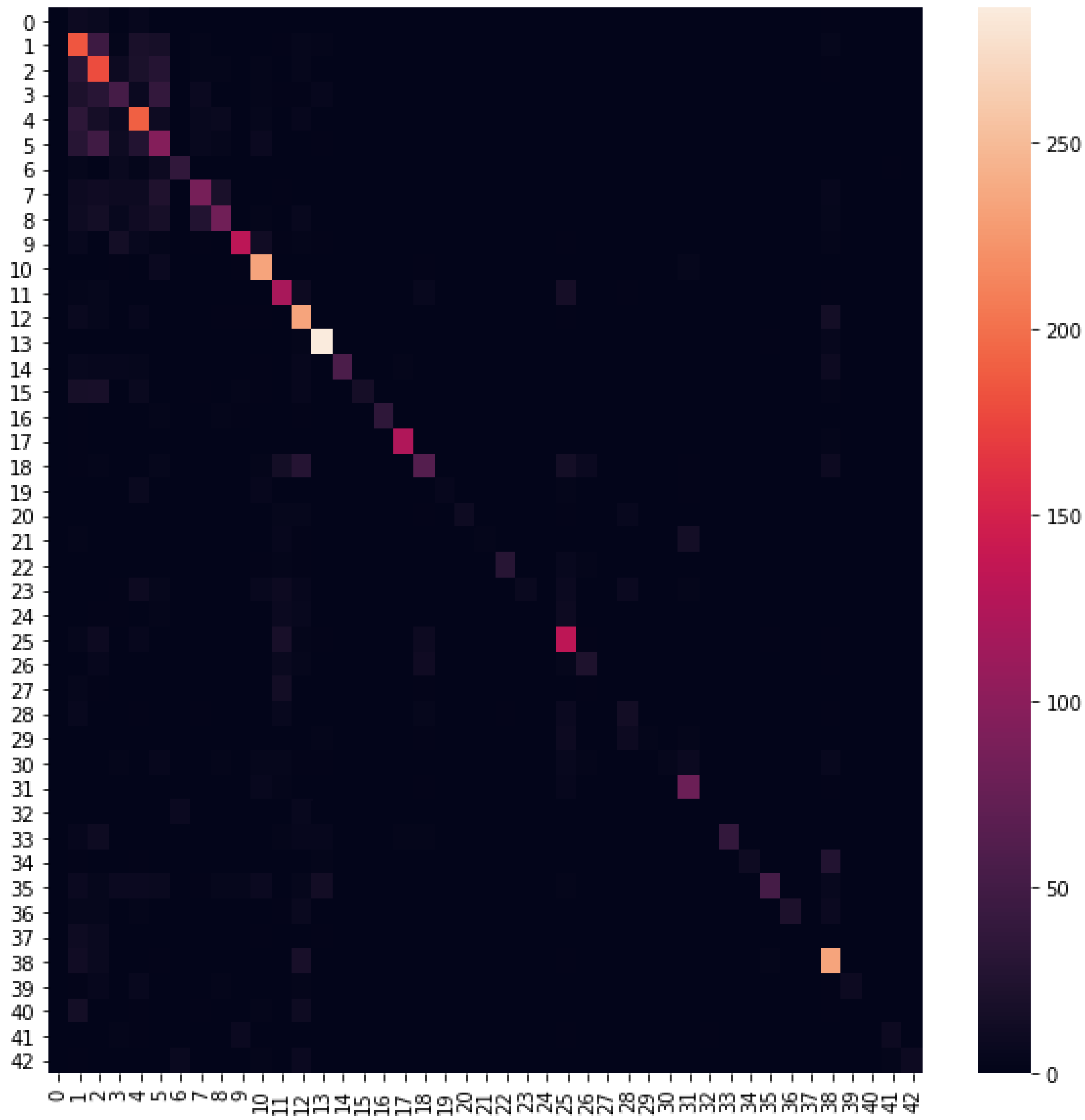
According to the Total Number of Training Images of Each Class ID bar graph and the Accuracy Score of Each Class ID with PCA Method Only bar graph, the higher the number of the training images of a traffic sign class ID as shown above, the better the accuracy score of a traffic sign class ID. Hence, one of the suggestions to improve the recognition rate of the PCA is to increase the number of training images of certain class IDs that have a lower percentage of correct classification. Besides that, increases the number of components of PCA or increase the PCA variance could help to improve the PCA recognition rate, for example, as shown in the result below, increase the PCA number of components by 32 has helped to improve the overall accuracy score by 1.2 %.

[33]:
```
# Select the number of components that contains 97% of the variance or 0.97␣
↪variance
variance_num = 0.97
# Create the PCA object.
pca = PCA(variance_num)
# Fit the flatten train images array
pca.fit(train_imgs_flatten)
# Print the total number of components need to get 97% of the variance
num_PCA_components = pca.n_components_
print("Number of Components Needed to Capture 97% of the Information: ",␣
↪num_PCA_components)

# Create the PCA object
pca = PCA(n_components = num_PCA_components)
# Fit and transform train images
x_train = pca.fit_transform(train_imgs_flatten)
# Transform test images
x_test = pca.transform(test_imgs_flatten)

# Create Random Forest Classifier Object
RF_classifier = RandomForestClassifier()

# Fit the flatten train images with the train IDs
RF_classifier.fit(x_train, train_IDs)

# Predict the test set outcomes.
y_predict = RF_classifier.predict(x_test)

# Plot Confusion Matrix
Confusion_Matrix = confusion_matrix(test_IDs, y_predict)
print("Confusion Matrix: \n\n", Confusion_Matrix)

# Show Heatmap
plt.figure(figsize=(10,10))
sns.heatmap(Confusion_Matrix)

# Print Classification Report
print("\n\nClassfication Report:␣
↪\n\n",classification_report(test_IDs,y_predict))

# Total Number of Test Images
total_num_test_imgs = np.sum(Confusion_Matrix)
# Total Number of Correct Classification
tot_num_correct_classification = np.trace(Confusion_Matrix)
# Total Number of Incorrect Classification
```

```
tot_num_incorrect_classification = total_num_test_imgs -␣
 ↪tot_num_correct_classification

# Print Recognition Rate Report
print('Total Number of Test Images: ', total_num_test_imgs)
print("Total Number of Correct Classifications: ",␣
 ↪tot_num_correct_classification)
print("Total Number of Incorrect Classifications: ",␣
 ↪tot_num_incorrect_classification)
print('\nAccuracy Score: ' + str(accuracy_score(test_IDs, y_predict)*100) + '␣
 ↪%')
```

```
Number of Components Needed to Capture 97% of the Information:  140
Confusion Matrix:

 [[   0   10    7 …    0    0    0]
 [   0 185   46 …    0    0    0]
 [   0   30 179 …    0    0    0]
 …
 [   0   15    1 …    0    0    0]
 [   0    0    0 …    0    9    0]
 [   0    2    0 …    0    3    9]]


Classfication Report:

              precision    recall  f1-score   support

           0       0.00      0.00      0.00        25
           1       0.40      0.64      0.49       291
           2       0.39      0.62      0.48       287
           3       0.34      0.30      0.32       174
           4       0.52      0.67      0.58       285
           5       0.33      0.41      0.36       234
           6       0.69      0.56      0.62        66
           7       0.55      0.47      0.51       185
           8       0.59      0.45      0.51       183
           9       0.80      0.69      0.74       191
          10       0.72      0.91      0.81       257
          11       0.51      0.72      0.59       165
          12       0.57      0.83      0.68       281
          13       0.83      0.96      0.89       299
          14       0.95      0.56      0.70       100
          15       0.94      0.20      0.33        80
          16       1.00      0.71      0.83        49
          17       0.90      0.92      0.91       136
          18       0.55      0.41      0.47       155
```

```
          19       0.83      0.18      0.29        28
          20       0.73      0.31      0.44        35
          21       1.00      0.09      0.17        33
          22       0.88      0.60      0.71        50
          23       0.64      0.10      0.18        68
          24       0.00      0.00      0.00        33
          25       0.54      0.69      0.61       194
          26       0.53      0.38      0.44        61
          27       0.00      0.00      0.00        25
          28       0.33      0.27      0.30        51
          29       1.00      0.09      0.17        33
          30       0.67      0.07      0.12        58
          31       0.66      0.75      0.70       105
          32       0.33      0.05      0.09        20
          33       0.97      0.49      0.66        79
          34       0.92      0.23      0.37        47
          35       0.83      0.39      0.53       132
          36       0.96      0.43      0.59        51
          37       0.00      0.00      0.00        26
          38       0.63      0.82      0.71       284
          39       1.00      0.26      0.41        39
          40       0.00      0.00      0.00        41
          41       0.60      0.31      0.41        29
          42       1.00      0.26      0.41        35

    accuracy                           0.58      5000
   macro avg       0.62      0.41      0.45      5000
weighted avg       0.61      0.58      0.56      5000

Total Number of Test Images:  5000
Total Number of Correct Classifications:  2907
Total Number of Incorrect Classifications:  2093

Accuracy Score: 58.14 %

/Users/kaitheuser/opt/anaconda3/lib/python3.7/site-
packages/sklearn/metrics/_classification.py:1272: UndefinedMetricWarning:
Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```



0.1.4 ii.) Combination of PCA and Linear Discriminant Analysis (LDA) Methods for Dimensionality Reduction with Random Forest Classification for Image Recognition

1.) Import all the libraries that will be used.

[19]:
```
# Import necessary libraries
import cv2
import math
import matplotlib
import matplotlib.pyplot as plt
import numpy as np
```

```
import pandas as pd
import seaborn as sns
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
from sklearn.decomposition import PCA
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.metrics import accuracy_score
```

2.) Load and store the "Train_subset.csv" and "Test_subset.csv" data as a dataframe.

[20]:
```
# Read the subset of the train data.
train_subset_df = pd.read_csv('Train_subset.csv')

# Read the subset of the test data.
test_subset_df = pd.read_csv('Test_subset.csv')
```

3.) Store training and testing images and Class ID into respective lists.

[21]:
```
# List that stores train images
train_imgs = []
# List that store train class ID
train_IDs = []
# List that stores test images
test_imgs = []
# List that store test class ID
test_IDs = []

# Size of the image,
img_size = 25

# Store train images into the train images list
for idx in range(0, len(train_subset_df)):

    # Load image
    img = cv2.imread(train_subset_df['Path'][idx], cv2.IMREAD_COLOR)
    # Resize the image to 25 x 25 because the minimum width of the images are␣
 ↪25 x 25
    img = cv2.resize(img, (img_size, img_size))
    # Convert the image to grayscale
    img = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
    # Add to train images list
    train_imgs.append(img)

    # Get the ClassID
    train_class_ID = train_subset_df['ClassId'][idx]
    # Add to train ID list
    train_IDs.append(train_class_ID)
```

```
# Store test images into the test images list
for idx in range(0, len(test_subset_df)):

    # Load image
    img = cv2.imread(test_subset_df['Path'][idx], cv2.IMREAD_COLOR)
    # Resize the image to 25 x 25 because the minimum width of the images are␣
↪25 x 25
    img = cv2.resize(img, (img_size, img_size))
    # Convert the image to grayscale
    img = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
    # Add to train images list
    test_imgs.append(img)

    # Get the ClassID
    test_class_ID = test_subset_df['ClassId'][idx]
    # Add to train ID list
    test_IDs.append(test_class_ID)

## Convert the lists to arrays
# Train images and IDs arrays
train_imgs = np.array(train_imgs)
train_IDs = np.array(train_IDs)
#train_IDs = train_IDs.reshape(train_IDs.shape[0], 1)
# Test images and IDs arrays
test_imgs = np.array(test_imgs)
test_IDs = np.array(test_IDs)
#test_IDs = test_IDs.reshape(test_IDs.shape[0], 1)

# Print to check the shape of the arrays
print("Train Image Array Shape: ", train_imgs.shape)
print("Train IDs Array Shape: ", train_IDs.shape)
print("Test Image Array Shape: ", test_imgs.shape)
print("Train IDs Array Shape: ", test_IDs.shape)

print("\n---------------------------------------------\n")

# Print the total number of different traffic signs
class_IDs = np.unique(train_IDs)
nClasses = len(class_IDs)
print("Total Number of Different Traffic Signs: ", nClasses)
print("Traffic Sign IDs: ", class_IDs)
```

```
Train Image Array Shape:  (10000, 25, 25)
Train IDs Array Shape:  (10000,)
Test Image Array Shape:  (5000, 25, 25)
Train IDs Array Shape:  (5000,)

---------------------------------------------
```

```
Total Number of Different Traffic Signs:  43
Traffic Sign IDs:  [ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19
20 21 22 23
 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42]
```

4.) Scale or normalize the train and test images pixel values in between 0 and 1.

[22]:
```python
# Normalize the train and test images pixel values.
train_imgs = train_imgs / 255.0
test_imgs = test_imgs / 255.0
```

5.) Reshape the train images dimensions from (Number of Images, N_{imgs} × Height of the Image, H × Width of the Image, W) to (N_{imgs} × HW).

[23]:
```python
# Flatten the train images.
train_imgs_flatten = train_imgs.flatten().reshape(train_imgs.shape[0],␣
 ↪img_size*img_size)

# Flatten the test images.
test_imgs_flatten = test_imgs.flatten().reshape(test_imgs.shape[0],␣
 ↪img_size*img_size)

print("Flatten Train Images Shape: ", train_imgs_flatten.shape)
print("Flatten Test Images Shape: ", test_imgs_flatten.shape)
```

```
Flatten Train Images Shape:  (10000, 625)
Flatten Test Images Shape:  (5000, 625)
```

6.) Perform PCA method with the sklearn library to compress data.

[24]:
```python
# Select the number of components that contains 96% of the variance or 0.96␣
 ↪variance
variance_num = 0.96
# Create the PCA object.
pca = PCA(variance_num)
# Fit the flatten train images array
pca.fit(train_imgs_flatten)
# Print the total number of components need to get 90% of the variance
num_PCA_components = pca.n_components_
print("Number of Components Needed to Capture 96% of the Information: ",␣
 ↪num_PCA_components)

# Create the PCA object
pca = PCA(n_components = num_PCA_components)
# Fit and transform train images
x_train = pca.fit_transform(train_imgs_flatten)
# Transform test images
x_test = pca.transform(test_imgs_flatten)
```

```
Number of Components Needed to Capture 96% of the Information:  108
```

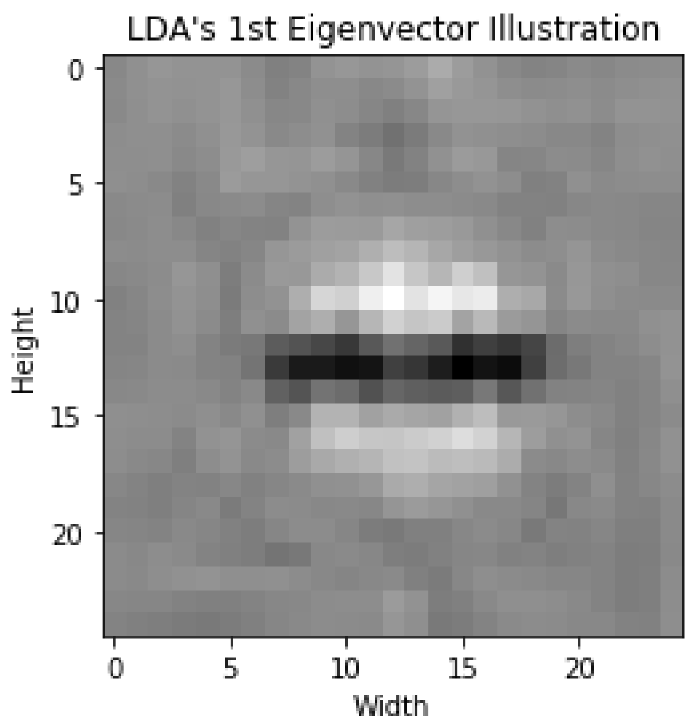
7.) Perform LDA method to maximize the between class separation.

[25]:
```python
# Create the LDA object
lda = LDA()
# Fit and transform train images
x_train = lda.fit_transform(x_train, train_IDs)
# Transform test images
x_test = lda.transform(x_test)
```

8.) Provide illustration of the first two eigenvectors of LDA by presenting the Fisherfaces.

[26]:
```python
# Get the first and second eigenvectors
first_eigenVector = lda.scalings_[:, 0]
second_eigenVector = lda.scalings_[:, 1]

# Get the Fisherface
Fisherface_1 = pca.inverse_transform(first_eigenVector)
Fisherface_2 = pca.inverse_transform(second_eigenVector)

# Illustrate the 1st Eigenvector of LDA
print("LDA's 1st Eigenvector:\n", first_eigenVector)
plt.title("LDA's 1st Eigenvector Illustration")
plt.xlabel("Width")
plt.ylabel("Height")
plt.imshow(Fisherface_1.reshape((img_size, img_size)), cmap ='gray')
plt.show()
```

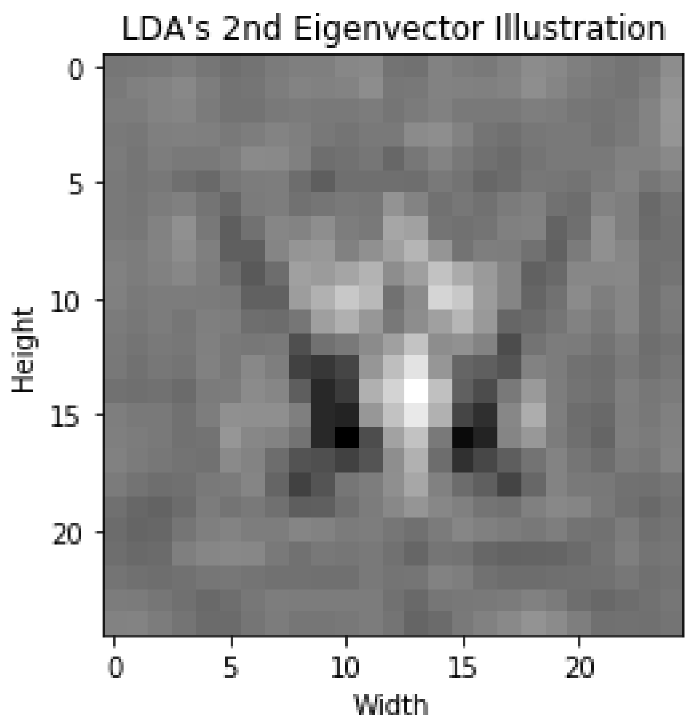
```
LDA's 1st Eigenvector:
 [ 0.00583374 -0.06850922 -0.05521733  0.00285296  0.31400536 -0.21233371
 -0.24589856  0.30502795  0.13663682  0.05066559  0.28449933 -0.63002975
  0.60974334 -0.94032512  2.02682844 -1.52692286  1.92999549 -0.1587126
  0.90745766 -0.84813311  0.7112735   0.28452083  0.14524421  0.10776787
 -0.02201662  0.26780417 -0.31551815  0.15864952 -0.03601579 -0.37755506
  0.23831951  0.80445416 -0.21636692 -0.31206182 -0.03112937 -0.67242164
 -0.38395553 -0.1371928  -0.46470449 -0.17523129 -0.51369271 -0.26906566
  0.22270884 -0.06607754  0.17540208  0.25075372  0.60814425 -0.00953622
 -0.35081487 -0.14861086 -0.6103466  -0.02608007  0.35394875  0.59060253
 -0.53579889 -0.25398956 -0.23204501 -0.05936088  0.15894516  0.17388726
 -0.18599854  0.34739729  0.136429   -0.14806789  0.35920694 -0.10849875
  0.0036123  -0.46834748 -0.11791117  0.11982869 -0.11734781 -0.00863435
  0.22310725 -0.19943186  0.23724138  0.16608492  0.02986398  0.07190105
 -0.23088042  0.05394038 -0.13675931  0.11508656 -0.10822236 -0.31039676
 -0.10016567 -0.10816218  0.0266321   0.20358032  0.08368979 -0.08713969
 -0.21274676  0.05070613 -0.12868585 -0.24044902 -0.09933473 -0.42407622
  0.14250069 -0.15582621 -0.249749   -0.10349418  0.13243247 -0.14731938
 -0.28336757 -0.21981142  0.0758956  -0.20578883  0.08653154 -0.20146625]
```



[27]:
```python
# Illustrate the 2nd Eigenvector of LDA
print("LDA's 2nd Eigenvector:\n", first_eigenVector)
plt.title("LDA's 2nd Eigenvector Illustration")
plt.xlabel("Width")
plt.ylabel("Height")
plt.imshow(Fisherface_2.reshape((img_size, img_size)), cmap ='gray')
plt.show()
```

```
LDA's 2nd Eigenvector:
 [ 0.00583374 -0.06850922 -0.05521733  0.00285296  0.31400536 -0.21233371
 -0.24589856  0.30502795  0.13663682  0.05066559  0.28449933 -0.63002975
  0.60974334 -0.94032512  2.02682844 -1.52692286  1.92999549 -0.1587126
  0.90745766 -0.84813311  0.7112735   0.28452083  0.14524421  0.10776787
 -0.02201662  0.26780417 -0.31551815  0.15864952 -0.03601579 -0.37755506
  0.23831951  0.80445416 -0.21636692 -0.31206182 -0.03112937 -0.67242164
 -0.38395553 -0.1371928  -0.46470449 -0.17523129 -0.51369271 -0.26906566
  0.22270884 -0.06607754  0.17540208  0.25075372  0.60814425 -0.00953622
 -0.35081487 -0.14861086 -0.6103466  -0.02608007  0.35394875  0.59060253
 -0.53579889 -0.25398956 -0.23204501 -0.05936088  0.15894516  0.17388726
 -0.18599854  0.34739729  0.136429   -0.14806789  0.35920694 -0.10849875
  0.0036123  -0.46834748 -0.11791117  0.11982869 -0.11734781 -0.00863435
  0.22310725 -0.19943186  0.23724138  0.16608492  0.02986398  0.07190105
 -0.23088042  0.05394038 -0.13675931  0.11508656 -0.10822236 -0.31039676
 -0.10016567 -0.10816218  0.0266321   0.20358032  0.08368979 -0.08713969
```

```
-0.21274676  0.05070613 -0.12868585 -0.24044902 -0.09933473 -0.42407622
 0.14250069 -0.15582621 -0.249749   -0.10349418  0.13243247 -0.14731938
-0.28336757 -0.21981142  0.0758956  -0.20578883  0.08653154 -0.20146625]
```



LDA Eigenvectors Explanation: The LDA Eigenvectors are similar to PCA Eigenvectors, but LDA eigenvectors maximize the separation between classes of two different traffic signs during the training process. Hence, PCA-LDA for traffic sign recognition is far more superior than the PCA traffic sign recognition. However, LDA is more noise senstive and create blurry effects on the images as illustrated above.

9.) Apply Random Forest Classification to perform image recognition.

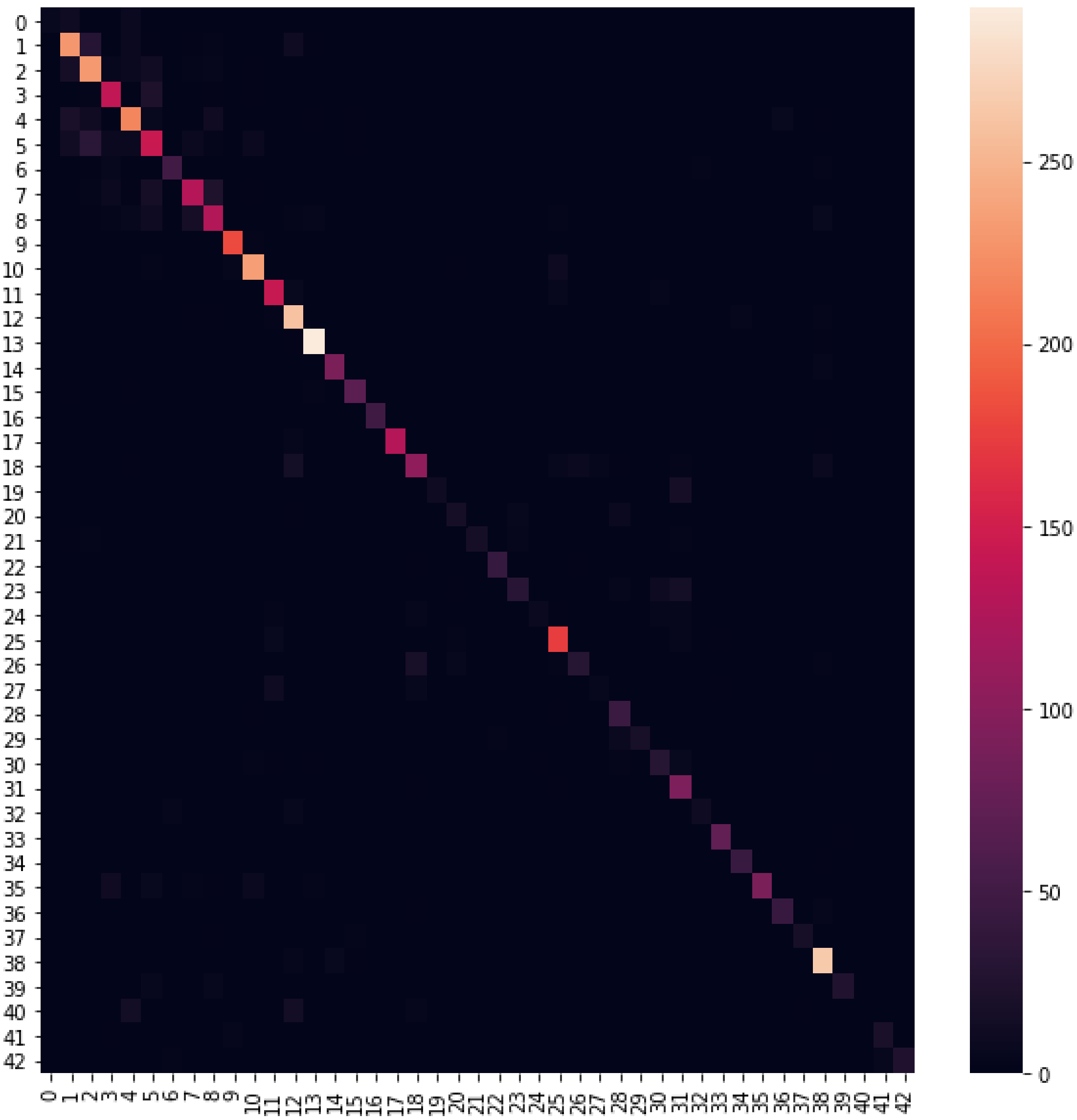
[28]:
```
# Create Random Forest Classifier Object
RF_classifier = RandomForestClassifier()

# Fit the flatten train images with the train IDs
RF_classifier.fit(x_train, train_IDs)

# Predict the test set outcomes.
y_predict = RF_classifier.predict(x_test)
```

10.) Compute the combination of PCA and LDA recognition rates for the test set to report correct classification and incorrect classification.

[29]:
```python
# Plot Confusion Matrix
Confusion_Matrix = confusion_matrix(test_IDs, y_predict)
print("Confusion Matrix: \n\n", Confusion_Matrix)

# Show Heatmap
plt.figure(figsize=(10,10))
sns.heatmap(Confusion_Matrix)

# Print Classification Report
print("\n\nClassfication Report:␣
↪\n\n",classification_report(test_IDs,y_predict))
```

```
Confusion Matrix:

 [[  6  11   0 …   0   0   0]
 [  0 230  29 …   0   0   0]
 [  0  17 231 …   0   0   0]
 …
 [  0   1   0 …   1   0   0]
 [  0   0   1 …   0  20   0]
 [  0   0   0 …   0   5  24]]


Classfication Report:

               precision    recall  f1-score   support

           0       1.00      0.24      0.39        25
           1       0.76      0.79      0.77       291
           2       0.71      0.80      0.76       287
           3       0.74      0.80      0.77       174
           4       0.79      0.76      0.78       285
           5       0.61      0.62      0.61       234
           6       0.85      0.77      0.81        66
           7       0.76      0.70      0.73       185
           8       0.67      0.70      0.69       183
           9       0.92      0.95      0.94       191
          10       0.87      0.91      0.89       257
          11       0.82      0.87      0.84       165
          12       0.76      0.93      0.84       281
          13       0.93      0.98      0.95       299
          14       0.89      0.92      0.91       100
          15       0.85      0.86      0.86        80
          16       0.98      0.98      0.98        49
          17       0.97      0.95      0.96       136
          18       0.71      0.68      0.69       155
          19       0.79      0.39      0.52        28
          20       0.50      0.46      0.48        35
```

21	1.00	0.48	0.65	33
22	0.89	0.84	0.87	50
23	0.76	0.46	0.57	68
24	0.57	0.24	0.34	33
25	0.81	0.90	0.85	194
26	0.70	0.49	0.58	61
27	0.50	0.20	0.29	25
28	0.63	0.88	0.74	51
29	0.86	0.58	0.69	33
30	0.59	0.52	0.55	58
31	0.63	0.89	0.74	105
32	0.73	0.55	0.63	20
33	0.86	0.94	0.90	79
34	0.86	0.94	0.90	47
35	0.94	0.70	0.80	132
36	0.84	0.84	0.84	51
37	0.90	0.69	0.78	26
38	0.86	0.94	0.90	284
39	0.79	0.67	0.72	39
40	0.50	0.02	0.05	41
41	0.80	0.69	0.74	29
42	0.96	0.69	0.80	35
accuracy			0.79	5000
macro avg	0.79	0.70	0.72	5000
weighted avg	0.79	0.79	0.79	5000

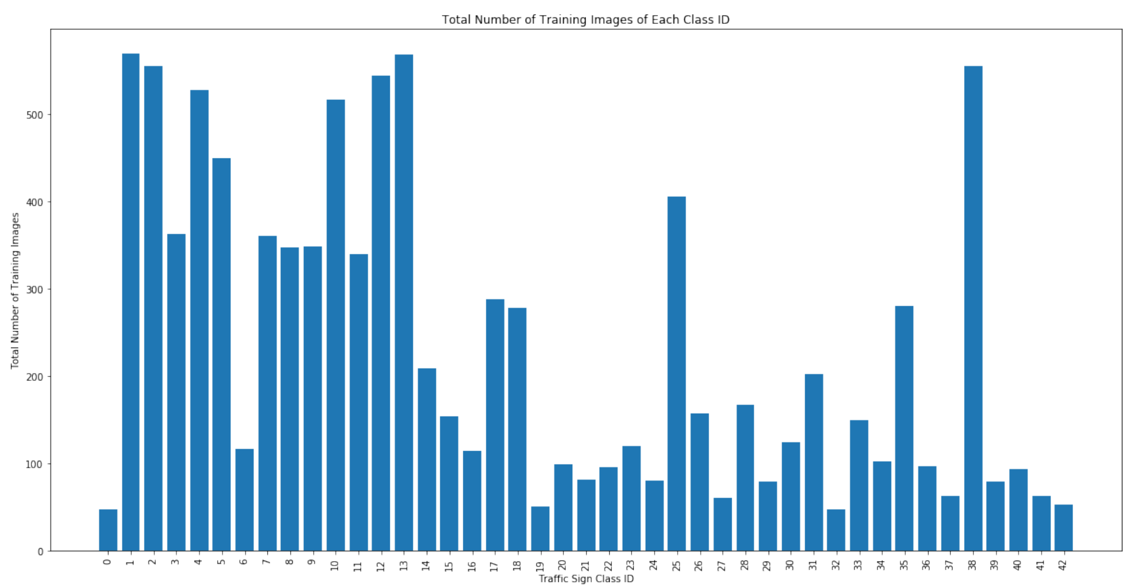


[30]:
```
# Total Number of Test Images
total_num_test_imgs = np.sum(Confusion_Matrix)
# Total Number of Correct Classification
tot_num_correct_classification = np.trace(Confusion_Matrix)
# Total Number of Incorrect Classification
tot_num_incorrect_classification = total_num_test_imgs -␣
↪tot_num_correct_classification

# Print Recognition Rate Report
print('Total Number of Test Images: ', total_num_test_imgs)
print("Total Number of Correct Classifications: ",␣
↪tot_num_correct_classification)
```

```
print("Total Number of Incorrect Classifications: ",␣
↪tot_num_incorrect_classification)
print('\nAccuracy Score: ' + str(accuracy_score(test_IDs, y_predict)*100) + '␣
↪%')
```

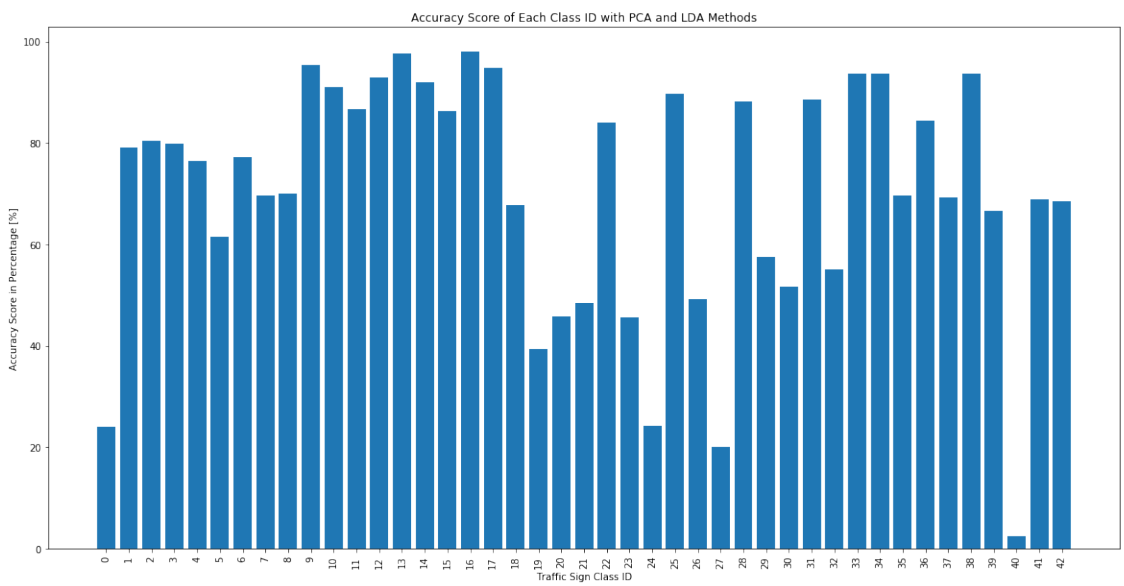
```
Total Number of Test Images:  5000
Total Number of Correct Classifications:  3970
Total Number of Incorrect Classifications:  1030

Accuracy Score: 79.4 %
```

[31]:
```
class_IDs_freq = np.zeros((1, nClasses))

# Store train images into the train images list
for idx in range(0, len(train_subset_df)):

    # Get the ClassID
    train_class_ID = train_subset_df['ClassId'][idx]
    # Count the numbers of different Class ID
    class_IDs_freq[0, train_class_ID] = class_IDs_freq[0, train_class_ID] + 1

plt.figure(figsize=(20,10))
plt.title('Total Number of Training Images of Each Class ID')
plt.ylabel('Total Number of Training Images')
plt.xlabel('Traffic Sign Class ID')
plt.bar(class_IDs, class_IDs_freq[0])
plt.xticks(class_IDs, rotation='vertical')
plt.show()
```



[32]:

```python
class_IDs_accuracy_score = np.zeros((1, nClasses))

# Calculate the correct classification percentage of each class
for idx in range(0, Confusion_Matrix.shape[0]):

    # Get the the sum of Class IDs
    sum_of_IDs = np.sum(Confusion_Matrix[idx,:])

    # Get the accuracy score in percentage
    class_IDs_accuracy_score[0, idx] = Confusion_Matrix[idx,idx] / sum_of_IDs *
 ↪100

plt.figure(figsize=(20,10))
plt.title('Accuracy Score of Each Class ID with PCA and LDA Methods')
plt.ylabel('Accuracy Score in Percentage [%]')
plt.xlabel('Traffic Sign Class ID')
plt.bar(class_IDs, class_IDs_accuracy_score[0])
plt.xticks(class_IDs, rotation='vertical')
plt.show()
```



Suggestion to Improve the PCA-LDA Recognition Rate

According to the Total Number of Training Images of Each Class ID bar graph and the Accuracy Score of Each Class ID with PCA-LDA Methods bar graph, the higher the number of the training images of a traffic sign class ID as shown above, the better the accuracy score of a traffic sign class ID. Hence, one of the suggestions to improve the recognition rate of the PCA-LDA is to increase the number of training images of certain class IDs that have a lower percentage of correct classification. Cropping the image to the region of interest would help to increase the recognition rate because it removes irrelevant features in the image. Overall, using the PCA-LDA methods for image recognition has a 22.46% higher recognition rate than the PCA method only for image recognition given that both have the same number of components, which is 108, and use the same image classifier, which is the Random Forest Classifier.

Contents


```
% Name          : Kai Chuen Tan
% Title         : Homework 4
% Course        : CSE 276C: Mathematics for Robotics
% Professor     : Dr. Henrik I. Christensen
% Date          : 13th November 2021

clear all;
clc; close all;

fprintf('Name           : Kai Chuen Tan\n')
fprintf('Title          : Homework 4\n')
fprintf('Course         : CSE 276C: Mathematics for Robotics\n')
fprintf('Professor      : Dr. Henrik I. Christensen\n')
fprintf('Date           : 13th November 2021\n\n')
fprintf('--------------------------------------------------------\n\n')
```

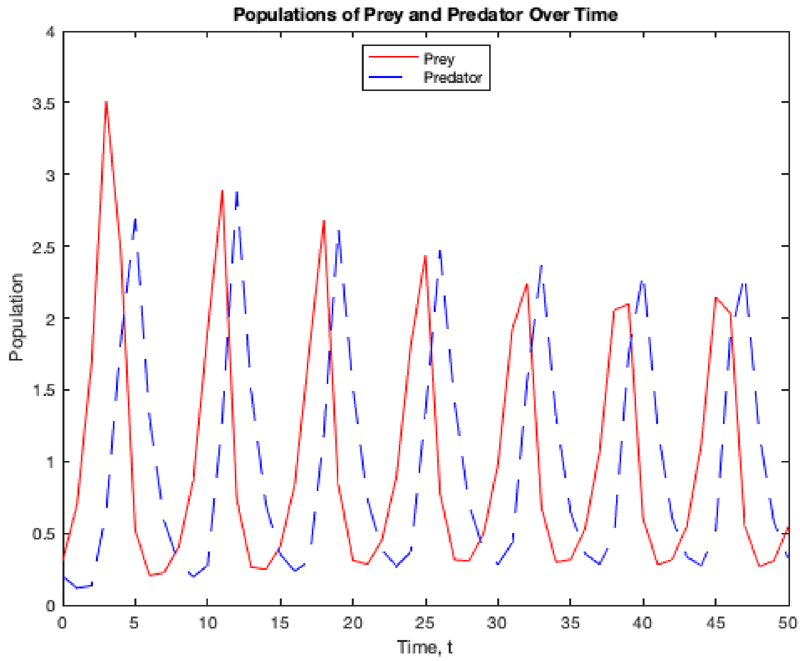
```
Name           : Kai Chuen Tan
Title          : Homework 4
Course         : CSE 276C: Mathematics for Robotics
Professor      : Dr. Henrik I. Christensen
Date           : 13th November 2021

--------------------------------------------------------
```

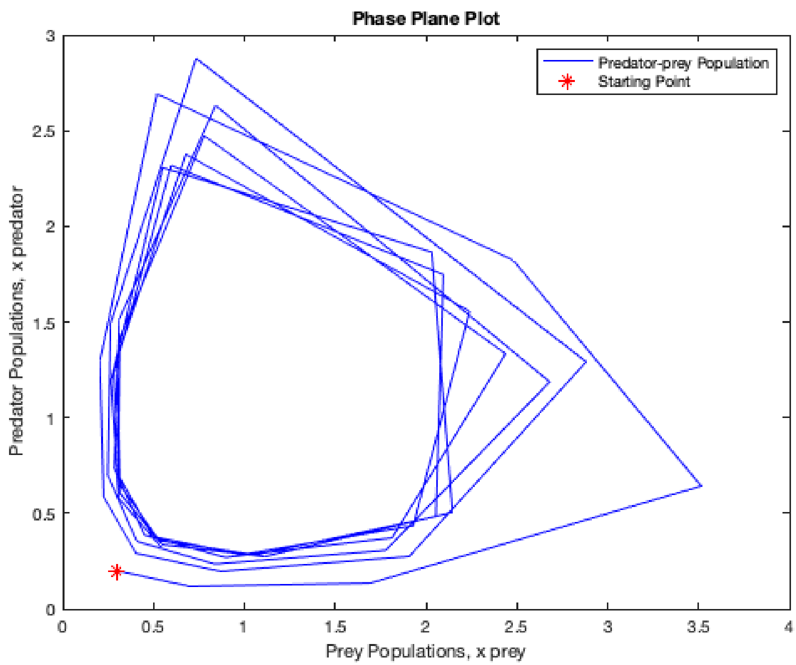
Problem 2 - Solving Predator-prey Model with Runge-Kutta

```
fprintf('Problem 2 - Solving Predator-prey Model with Runge-Kutta\n')
fprintf('--------------------------------------------------------------------\n\n')

% Initialize the coefficients
b = 1; p = 1; r = 1; d = 1;

% Time interval
t0 = 0; % Initial Time
t_final = 50; % Final Time

% Step size
h = 1;

% Number of intervals
n = (t_final - t0) / h;

% Time Array (1 x n+1)
t = zeros(1, n+1);
t(1) = t0;

% Prey array, x1 (1 x n+1)
x_prey = zeros(1, n+1);
x_prey(1) = 0.3;

% Predator array, x2 (1 x n+1)
x_predator = zeros(1, n+1);
x_predator(1) = 0.2;

% Define Lotka-Volterra Predator-prey Model.
% x1 - Prey Population
% x2 - Predator Population
```

```
dx1_dt = @(x1, x2) (b - p * x2) * x1;
dx2_dt = @(x1, x2) (r * x1 - d) * x2;

% Runge-Kutta Fourth Order
[t, x_prey, x_predator] = ODE_Runge_Kutta_4(dx1_dt, dx2_dt, t, x_prey, x_predator, h, n);

% Plot Populations Over Time and Phase Plane
plot_Populations_Over_Time(t,x_prey, x_predator)
plot_Phase_Plane(x_prey,x_predator)

% Print Comment on the Results
fprintf("Figure 2 - Phase Plane Plot illustrates that it somewhat converges to an orbit with a step size of 1,\n")
fprintf("but due to the large global error, O(h^5), it does not converge back to the initial point. Hence, the step\n")
fprintf("size will adjusted to a smaller step size, which is 0.1.\n\n")
```

```
Problem 2 - Solving Predator-prey Model with Runge-Kutta
--------------------------------------------------------------------

Figure 2 - Phase Plane Plot illustrates that it somewhat converges to an orbit with a step size of 1,
but due to the large global error, O(h^5), it does not converge back to the initial point. Hence, the step
size will adjusted to a smaller step size, which is 0.1.
```





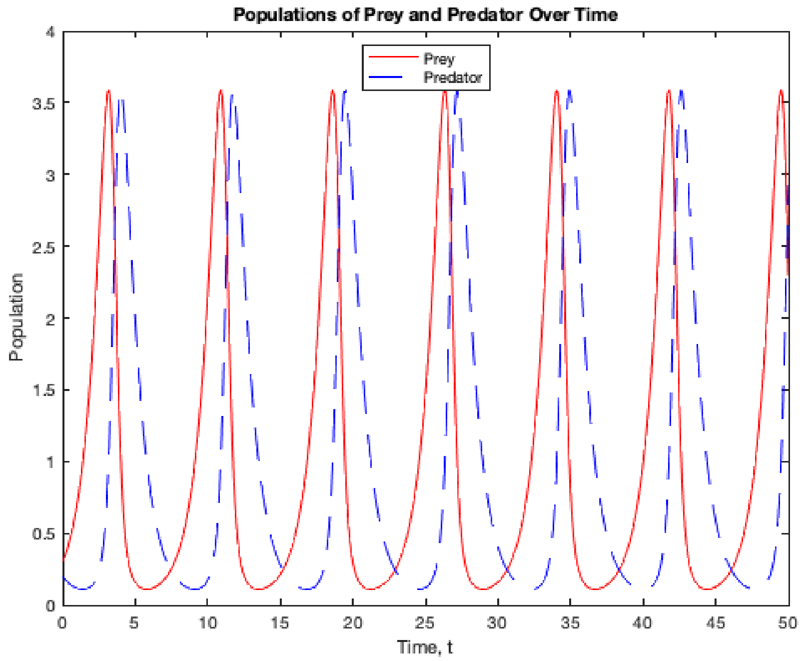
Smaller Step Size, h = 0.1

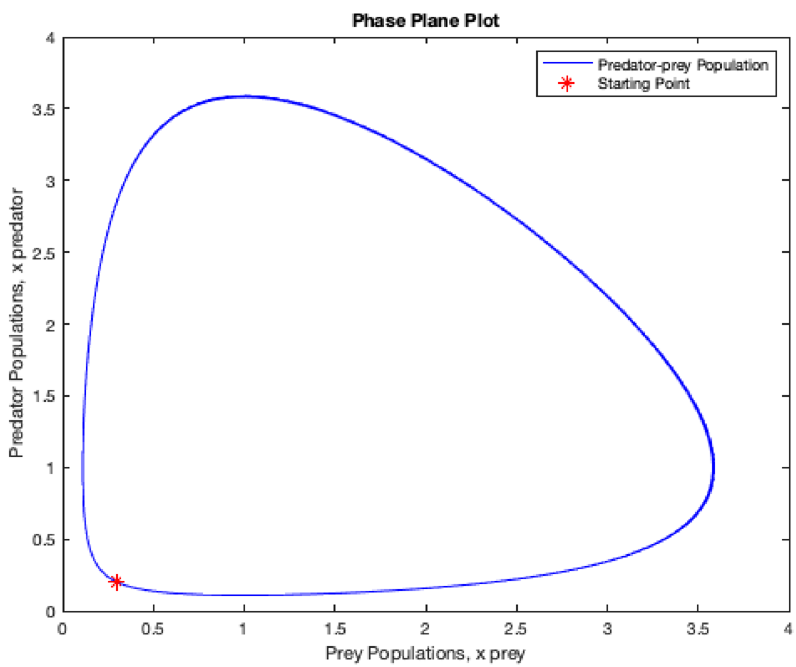
Step size

```
h = 0.1;

% Number of intervals
n = (t_final - t0) / h;

% Time Array (1 x n+1)
t = zeros(1, n+1);
t(1) = t0;

% Prey array, x1 (1 x n+1)
x_prey = zeros(1, n+1);
```

```
x_prey(1) = 0.3;

% Predator array, x2 (1 x n+1)
x_predator = zeros(1, n+1);
x_predator(1) = 0.2;

% Runge-Kutta Fourth Order
[t, x_prey, x_predator] = ODE_Runge_Kutta_4(dx1_dt, dx2_dt, t, x_prey, x_predator, h, n);

% Plot Populations Over Time and Phase Plane
plot_Populations_Over_Time(t,x_prey, x_predator)
plot_Phase_Plane(x_prey,x_predator)

% Print Comment on the Results
fprintf("Figure 4 - Phase Plane Plot illustrates that it converges to an orbit with a step size of 0.1,\n")
fprintf("Figure 3 - Populations of Prey and Predator Over Time presents that as the prey population increases,\n")
fprintf("the predator population started to rise exponentially due to the increase in foods. However, as the \n")
fprintf("predator increases exponentially, the prey population starts to drop drastically. Because of the drastic\n")
fprintf("drop in the prey population, the predator population starts to drop as well due to insufficient food.\n")
fprintf("The drop in prey and predator populations continues until the prey and predator populations reach\n")
fprintf("the initial prey population value and the initial predator population value as shown in the Figure 4 -\n")
fprintf("Phase Plane Plot. Then, the cycle will repeat.\n")
```

```
Figure 4 - Phase Plane Plot illustrates that it converges to an orbit with a step size of 0.1,
Figure 3 - Populations of Prey and Predator Over Time presents that as the prey population increases,
the predator population started to rise exponentially due to the increase in foods. However, as the
predator increases exponentially, the prey population starts to drop drastically. Because of the drastic
drop in the prey population, the predator population starts to drop as well due to insufficient food.
The drop in prey and predator populations continues until the prey and predator populations reach
the initial prey population value and the initial predator population value as shown in the Figure 4 -
Phase Plane Plot. Then, the cycle will repeat.
```





Published with MATLAB® R2021a

```
function [t, x_prey, x_predator] = ODE_Runge_Kutta_4(dx1_dt, dx2_dt, t, x_prey,↙
x_predator, h, n)
% ODE_Runge_Kutta_4 solves 1st order initial value ODE with Runge-Kutta
% Fourth Order's Method
% dx1_dt     - First Order Differential Equation of the Prey Population
%              Over Time
% dx2_dt     - First Order Differential Equation of the Predator Population
%              Over Time
% t          - Time Array
% x_prey     - Prey Array
% x_predator - Predator Array
% h          - Step Size
% n          - Number of Intervals

% Apply Runge-Kutta Fourth Order.
for k = 1 : n

    % Update time array, t
    t(k+1) = t(k) + h;

    % Update prey array, x_prey, and predator array, x_predator
    K_1_prey = dx1_dt(x_prey(k), x_predator(k));
    K_1_predator = dx2_dt(x_prey(k), x_predator(k));

    x_prey_K1 = x_prey(k) + K_1_prey / 2 * h;
    x_predator_K1 = x_predator(k) + K_1_predator / 2 * h;

    K_2_prey = dx1_dt(x_prey_K1, x_predator_K1);
    K_2_predator = dx2_dt(x_prey_K1, x_predator_K1);

    x_prey_K2 = x_prey(k) + K_2_prey / 2 * h;
    x_predator_K2 = x_predator(k) + K_2_predator / 2 * h;

    K_3_prey = dx1_dt(x_prey_K2, x_predator_K2);
    K_3_predator = dx2_dt(x_prey_K2, x_predator_K2);

    x_prey_K3 = x_prey(k) + K_3_prey * h;
    x_predator_K3 = x_predator(k) + K_3_predator * h;

    K_4_prey = dx1_dt(x_prey_K3, x_predator_K3);
    K_4_predator = dx2_dt(x_prey_K3, x_predator_K3);

    x_prey(k+1) = x_prey(k) + h / 6 * (K_1_prey + 2 * K_2_prey + 2 * K_3_prey +↙
K_4_prey);
    x_predator(k+1) = x_predator(k) + h / 6 * (K_1_predator + 2 * K_2_predator + 2 *↙
K_3_predator + K_4_predator);

end
end
```

```
function [] = plot_Populations_Over_Time(t,x_prey, x_predator)
% Plot Predator-prey Population Over Time
% t - Time array
% x_prey - Prey Array
% x_predator - Predator Array


figure;

% Plot the graph
plot(t, x_prey, 'r-', t, x_predator, 'b--')
% Include plot title
title('Populations of Prey and Predator Over Time')
% Label x-axis
xlabel('Time, t')
xlim([0 50])
% Label y-axis
ylabel('Population')
% Include Legends
legend('Prey', 'Predator', 'Location', 'North')

end
```

```matlab
function [] = plot_Phase_Plane(x_prey,x_predator)
% Plot Phase Plane Plot (Predator vs Prey)
% x_prey - Prey Array
% x_predator - Predator Array


figure;

% Plot the graph and the starting point
plot(x_prey, x_predator, 'b-', x_prey(1), x_predator(1), 'r*')
% Include plot title
title('Phase Plane Plot')
% Label x-axis
xlabel('Prey Populations, x prey')
% Label y-axis
ylabel('Predator Populations, x predator')
% Include Legends
legend('Predator-prey Population', 'Starting Point', 'Location', 'Northeast')

end
```