# Where's Waldo? A Detailed Analysis on How to Find Waldo Via Computer

Katrina Rhone, Santino Macaggi, Tanisha Konduru, and Hayden Corrado

Department of Engineering, Purdue University

ENG 133: First Year Engineering

October 18, 2022

# Where's Waldo? A Detailed Analysis on How to Find Waldo Via Computer

## Introduction

Object/feature detection is a method to find a desired aspect of something within a large picture. These aspects include, but are not limited to people, cars, license plates, and animals (Boesch 2022). But object detection can be used to find anything that the user wants within the picture if it is programmed to do so. Group 15 researched object/feature detection by looking at websites, what information was given to us in class, and by asking questions during office hours. Our group used the information we gathered throughout these past two weeks and applied person (feature) detection in our first template of Where's Waldo by importing our image and desired template and gray-scaling them, running them through the three different ways of doing object detection, and then using a main function to call each algorithm. Once our program runs, it outputs a gray-scaled version of our original image, our original image with a green rectangle around the template image we were trying to find, and a cropped image of the rectangle around the template image.

## Project Motivation

Object/feature detection is widely utilized in many fields of work. For instance, our group researched how object detection is implemented in the field of mechanical engineering. So, how does a mechanical engineer use object/feature detection? Mechanical engineers use this resource when they are trying to develop a "semiautonomous safety system" (Chu, 2012). In this article by Chu, she is depicting a car with a system (machine) in it to help the driver and car stay safe and

alert while driving on the road (2012). Object detection is essential in having a system recognize potential hazards on the road, street signs, and other cars around the automobile with the safety system in it (Chu, 2012).

Object/feature detection is also being developed to detect defects in manufacturing. At Old Dominion University, Juan C. Parducci worked on a study that designed and developed an object detector that would be able to scan and identify defects in the production of rims and tires (Parducci, 2022). As of right now, quality control for these products is being done manually by quality control specialists. This method of inspection and quality control is not only time consuming, but also has a much higher margin of error. While some defects in products are larger and more evident, others may be smaller and more difficult to identify quickly by human inspection (Parducci, 2022). Implementing object/feature detection in this practice would be significantly more feasible and ultimately increase the Six Sigma certification and rating of the manufacturer as a whole.

Fault Detection and Isolation is a very important part of manufacturing and mechanical construction. In its current state, objects, such as nuts and bolts, need to be manually checked to ensure that they match the required specifications. However, human checks are done randomly and have the ability to miss imperfect parts. There needs to be a better way of doing fault detection and Isolation. C. Bharathi Priya and V.Sudha, two Ph. D students at the College of Engineering, Guindy in India, have come up with a better way to do this process, making it more accurate by eliminating the human element. In their proposal, they use a camera on the conveyer belt to take pictures and scan each image to see if any nuts or bolts have imperfections: “an electrical signal will be sent to the Solenoid valve and then it actuates deflector plate by the pneumatic cylinder.” (C. Bharathi Priya and V.Sudha). This can be used to improve a job done by a human.

## Project Overview and Methods

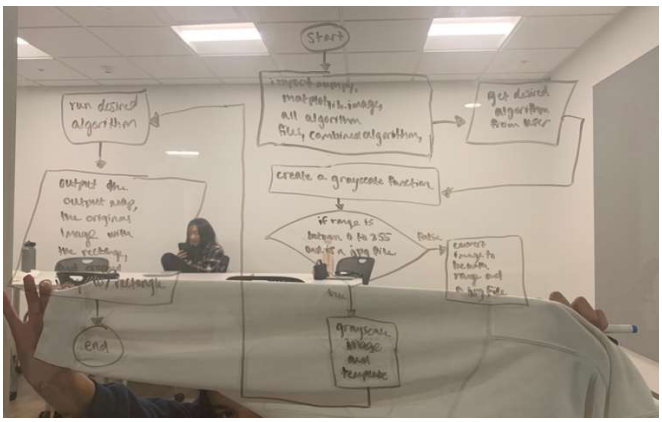
The background behind our template matching to an image starts with gray-scaling the desired image and then putting it in a matrix format for the other three algorithms to use. You first gray scale before doing the object detection because it takes away the complexity of dealing with different colors in various images and templates. In this way, the computer can focus on finding its matching matrices that is desired. The Sum of Squared Differences equation takes the pixels of the original image and the template image and compares them to find a point at which the difference between them has the smallest value. The coordinates of this value is the approximation of the location of the template on the original image. Coefficient Correlation is a strategy that can be used for image detection. For this strategy, one must import an image and an image template, convert it to grayscale, and then make an image map matrix from the image. The first actual step of the CC strategy is to find the average values of the template matrix. Once that is determined, that average-value matrix is subtracted from the template image's matrix, giving what we define as T'. Then, you find the next step by using a "window" to look at the image matrix, find the average of each window, and subtract them from the window matrix. This is what we defined as I'. Finally, for every I and J value in the output map you find the sum of I' multiplied by T'. The max value in this new R matrix is the top left corner of the template image. The Normalized Cross-Correlation algorithm is a method that can be used for template matching. This algorithm takes in an image and an image template. The algorithm compares the template image to a specific part of the original image. This is done by looping through each entry in the image and comparing the section of that entry to the template. Every time the comparison is done, a calculated value is appended to another matrix. By the end, a matrix of all the comparisons will be made. The section

of the image that most closely resembles the template will be found by finding the maximum value of that new matrix. The coordinates of this value can show where the image most closely resembles the template.

## Discussion of Algorithm Design

**Figure 1**

*Flowchart for the main program*

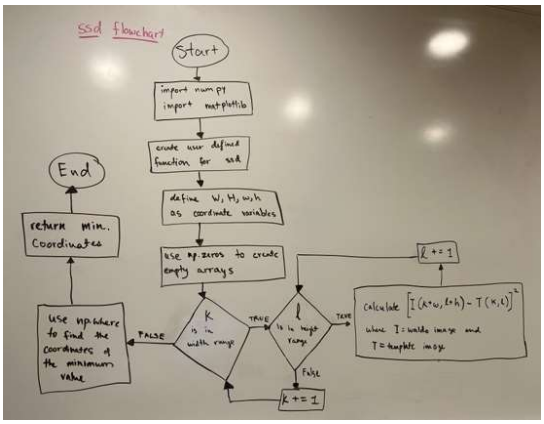



The main program was designed so that future images and templates would also be able to use the object detection we have created. Group 15 did this by adding an input statement where the user inputs which files, they would like to have a specific object or feature detected and then the function will change the images to the correct file type and run it through which algorithm would work best with the matrices of the image and template that were given. The main function algorithm starts with the user giving which image they would like to start with, as well as a template of what the user wants to find within the image. Then it ties the Sum of Squared

Differences, Cross-Correlation and Normalized Cross-Correlation for the file to go through. Then, as the final product, the user will see where in the image the object/feature that they were looking for is.

**Figure 2**

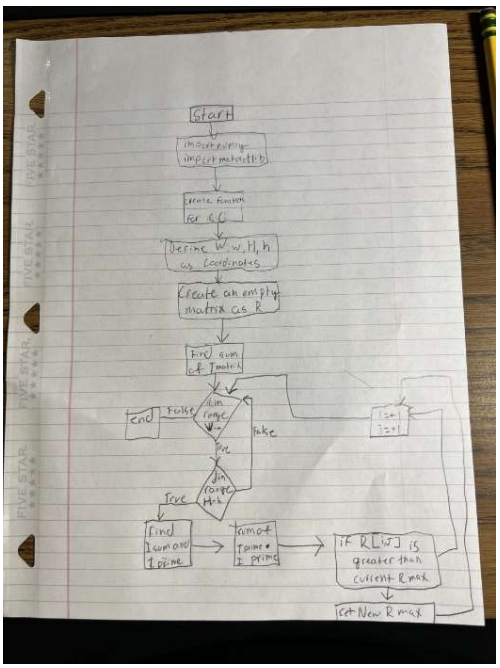
*Flowchart for Sum of Squared Differences*




The sum of squared difference's function was designed to take inputs of an image and a desired template image and compare matrices representing the pixel values of each image to calculate the difference between them. The function first imports the "numpy" and "matplotlib" modules to use later in the program to create empty arrays and find the coordinates of the minimum difference once both images are compared (Numpy User Guide). It defines the variables "H", "W", "h", and "w" to use later as the ranges of each image that the "for loops" run through. It then uses those ranges and the inputted image and template image in the sum of squared differences equation to calculate the difference. It finally returns the coordinates of the point where the difference between the template image and original image is the least representing an approximation for the location of waldo. This returned coordinate value is later used in the main program when the SSD function is called to find waldo.

**Figure 3**

*Flowchart for Correlation Coefficient*

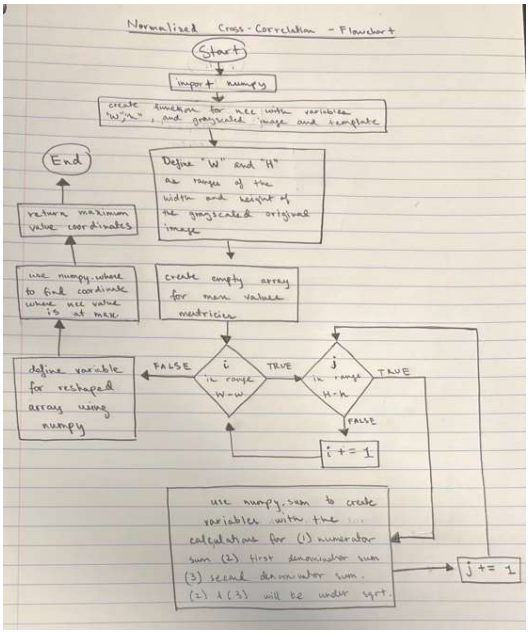



The Correlation Coefficient function was designed to take an input matrix of an image and a template from the image and find the location that best matches the pixel values of the image. To do this one must input numpy and matplotlib which will be used for the calculations and to read in the images (Numpy User Guide). From there the program defines W, w, H, h as the width and height of the main image and the template image respectively. From there those variables will be used as a range in a for loop to find the I resultant. However, before the I resultant is found the program will use np.sum to find the sum and then the average of the template matrix. To find the I resultant one will make two for loops for i and j, indexes on the output map, to find the sum of the "window" on the main image, then move it over one pixel each time it loops. From there I resultant can be found which then the code uses to find the indexes for the output map matrix. This

can be used for future projects as the code can read in any image and do this process for that given image. However, it does not always work by itself and may have to be used along with another image detection strategy to find the precise location of the template in the image.

**Figure 4**

*Flowchart for Normalized Cross-Correlation*




The Normalized Cross-Correlation function was designed to take an input image and a template from the image. The algorithm finds a location on the image that best matches the template image. This is done through comparing the matrix of the template to a part of the matrix of the image. To complete this program, NumPy and matplotlib will first need to be imported (Numpy User Guide). The algorithm loops through each entry in the image matrix that is grey scaled. For each entry, the program will compare the template matrix with a certain area of the image determined by the current entry. This is done through various summations. After the summations are completed for

one entry, that resulting calculation is added to a matrix. After the summations are completed for the entire image, a complete resulting matrix will be created. This is the matrix that is plotted using matplotlib. By finding the maximum value in that matrix, the coordinates of the specific object in the image can be found. This algorithm can be used in the future since it can work for other images besides the Waldo image. However, all three algorithms might need to be combined for some images.

## Summary of Discussion of Algorithm Design

The Sum of Squared Differences function finds the difference between the matrices of both images and identifies the point where the difference between them is the least. This minimum coordinate value approximates the location of the template. Meanwhile, the Cross Correlation and Normalized Cross Correlation functions identify the point where the difference is the maximum and use that point to locate the template image. Sum of squared differences took 87 seconds. Cross-correlation took 192 seconds. Normalized cross correlation took 267 seconds. Finally, for the function that combines all the algorithms, took 577 seconds. For our real-world application image and template, we got the image from Google and then used the snipping tool to get the portion of the image we wanted to find. Our image that we selected was of Finding Nemo. We were interested in this image specifically because Hayden thought it was interesting how Finding Nemo was so similar to Where's Waldo. Except Finding Nemo does not have the same connotation to it that Waldo does, so we wanted our image to create a similar connotation for Finding Nemo that Where's Waldo has. Each method performed well because Waldo was found, the outputted images were clear, and the box surrounded Waldo with each output for each

algorithm. The methods are performed like this because our main function zooms in the picture to find waldo at the desired matching matrices.

## References

Boesch, G. (2022). Object Detection in 2022: The Definitive Guide. *Viso.ai.* Object Detection in 2022: The Definitive Guide - viso.ai

Chu, J. (2012). Mechanical engineers develop an 'intelligent co-pilot' for cars. *Massachusetts Institute of Technology.* Mechanical engineers develop an 'intelligent co-pilot' for cars | MIT News | Massachusetts Institute of Technology

"Numpy User Guide." *Numpy User Guide – ENGR 133*, Purdue University, 26 May 2016, https://engineering.purdue.edu/fye_i2i/i2i/engr-133/helpful-course-documents/numpy-user-guide/.

Parducci, J. C. (2022). Deep Learning Object-Based Detection of Manufacturing Defects in X-Ray Inspection Imaging. *ODU Digital Commons*, Old Dominion University https://digitalcommons.odu.edu/cgi/viewcontent.cgi?article=1343&context=mae_etds

Priya, G. C.(2019).Image Processing Based Fault Detection and Isolation for Mechanical Components. *International Journal of Engineering and Advanced Technology.* https://www.ijeat.org/wp-content/uploads/papers/v8i6S/F10200886S19.pdf

# Appendix A

## User Manual

We made this program relatively easy to comprehend for people who are somewhat familiar with python. To start you, the user is prompted with a statement that tells them to input the name of the image that they want to use, and then the name of the template image that they want to use to find an object within that image. Lastly, the user must input what kind of algorithm they want the image to be ran through. Their options for the algorithm include: Sum of Squared Difference (“SSD”), Cross-Correlation(“CC”), Normalized Cross-Correlation(“NCC”), or you have the option to run all the algorithms(“ALL”).

**Figure A1**

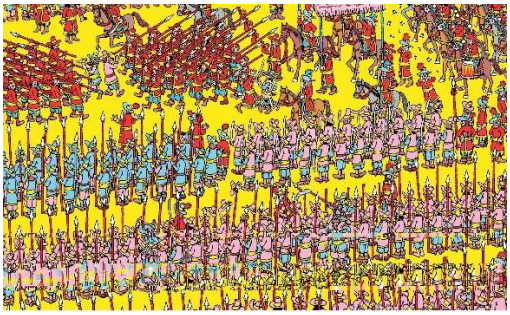


*Figure A1 is the Waldo image the user inputs into the main function*

**Figure A2**

*Figure A2 is the Waldo template the user inputs into the main function*

**Figure A3**

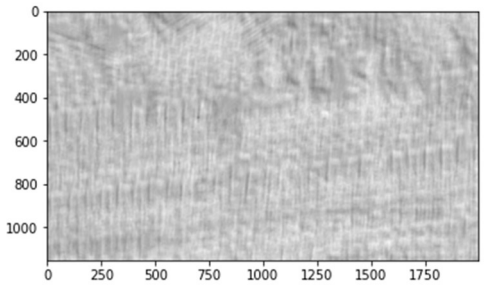



*Figure A3 is the Waldo image gray-scaled from the sum of squared difference's function*

**Figure A4**

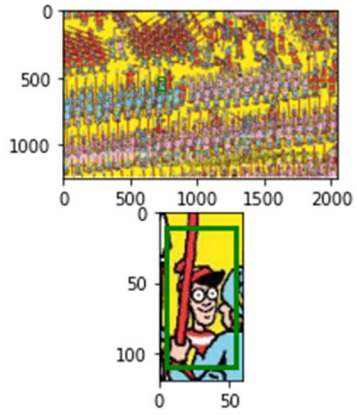



*Figure A4 is the sum of squared differences output of what was inputted by the user*

**Figure A5**

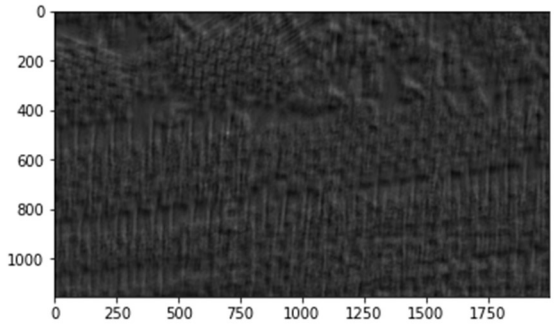



*Figure A5 is the normalized cross correlation gray-scaled of Waldo image*

**Figure A6**

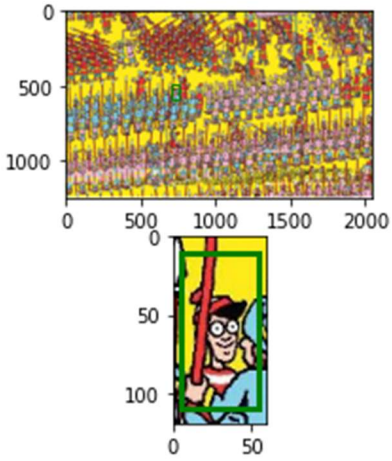



*Figure A6 is the normalized cross correlation output of Waldo being found*

**Figure A7**

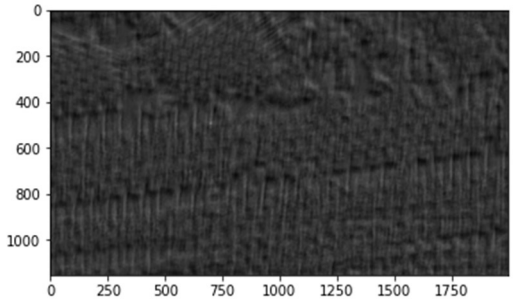



*Figure A7 is the cross-correlation grayscale of inputted Waldo image*

**Figure A8**

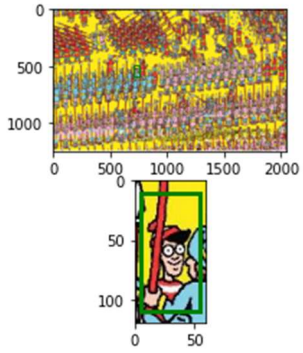



*Figure A8 is the cross-correlation output of Waldo being found*

**Figure A9**

*Figure A9 is the Finding Nemo image that we use for our real application image*

**Figure A10**

*Figure A10 is the Finding Nemo template that we use for our real application*

**Figure A11**

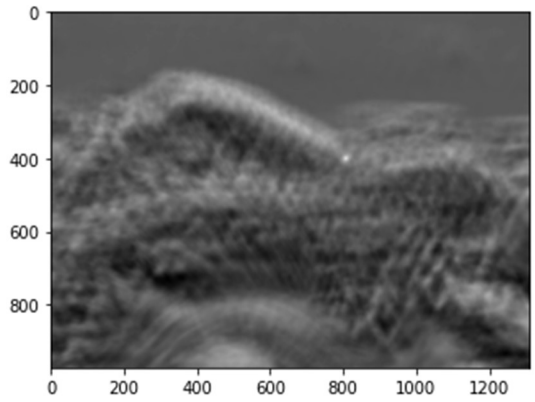



*Figure A11 is all the algorithm's combined grayscale of the Finding Nemo image*

**Figure A12**

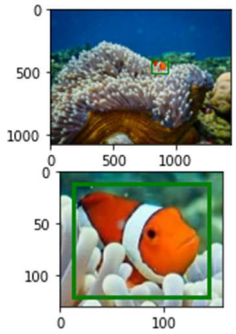


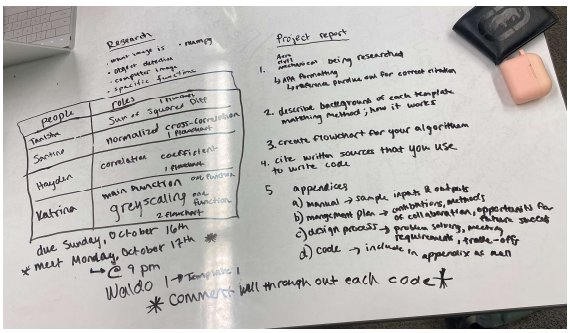

*Figure A12 is all the algorithm's combined output of Nemo being found*

## Appendix B

## Project Management Plan

Due to this being a lengthy project, we divided up the workload amongst one another and then met up to compare and ask for help from other group members. Katrina's assignment was to do the gray-scaling with the main function combining all the algorithms together. Additionally, to structure the report in a way that was easy to enter in what everyone did in APA format. Santino was assigned the normalized cross-correlation algorithm and making sure the codes worked with the real-life application. Santino was also the group's go to for help when trying to figure out how to get our codes to work. Tanisha worked on the sum of squared differences portion of the project; leaving Hayden to work on the correlation coefficient. Our team facilitated participation by coming to class after fall break having an idea of what we wanted our code to look like and bouncing ideas off each other. We also, had a game plan before we went on fall break of when we wanted to have things done before the actual deadline.

**Figure B1**




*Projection plan before the start of project*

Figure B1 shows our written-out pre-plan for the project that our group developed when we got this project as an assignment. As with most projects, things do not always go swiftly. For our project of creating object detection, we did not start working on the project when we first got

assigned the material. So, if we were to do it again, we should start when we first are assigned the project. Another improvement which could have been made is having our group go to more of the office hours that were held for us. We did manage to make it to one office hours but going to more maybe would have simplified our code more.

## Appendix C

## Discussion of Design Process

For our design process, Team 15 made an outline/timeline to meet the requirements for this project, with some wiggle room for mistakes if something were to go wrong. For instance, our goal was to have all our divided-up assignments were done on Sunday, October 16th. This insured that by Sunday we started on typing up the report, and each team member could input what they did into the report. Furthermore, we discussed that we were going to have our last meeting to fully finalize everything we had on Monday, October 17th. Monday night is also when we wanted to submit the entirety of the project by, so that we were prepared to do a demonstration for class on Tuesday morning. Then, before each member even began creating their code, we assigned each member individually to research what an image is, object detection, and specific functions that help with object detection. We felt that this would give us a better understanding of project before we even started so that we could be the requirements listed under the various constraints of the project we had. For the design process in the code, we had to rework the main function several times to be able to get the code to run smoothly. This was at times frustrating, but as a team we came together to put our brains together and get the synthesis of our codes up and running. We made some trade-offs when we first started the project with how we assigned who was doing what. The group members who are not the best at coding and have not done it a lot before picked which assignment they thought that they could handle. Meanwhile, the group members who are more confident in their codes were more optimistic about handling whichever code was left. Overall, with the help of our design process we think the project went smoothly.

**Appendix D**

**Code**

## Main Function Code:

```
import numpy as np

import matplotlib.image as img

from PythonTeam15ProjectNormCCORR import output_ncc as ncc

from PyTeam15ProjectSSD import output_ssd as ssd

from combining_algorithm import combined_algorithm

from PyTeam15ProjectCCAlgorithm import output_cc as cc

import time

#Each algorithm(SSD,CC,NCC, and the combined algorithm) is imported into this main file



def grayscale(colored_image, colored_template):



    matrix_image = np.array(colored_image)
```

```
    matrix_template = np.array(colored_template)
    #This turns the inputed image and template into matrices


    if color_image_input.split('.')[1] != 'jpg': #making sure image is a jpg file
        matrix_image = np.round(np.dot(matrix_image, 255/matrix_image.max()))
    if color_template_input.split('.')[1] != 'jpg': #making sure template is a jpg file
        matrix_template = np.round(np.dot(matrix_template, 255/matrix_template.max()))
    #These if statements make sure that the inputs are the correct datatype and are jpg files
    matrix_image_grey = np.dot(matrix_image[:,:,:3],[0.299, 0.587, 0.114])
    matrix_template_grey = np.dot(matrix_template[:,:,:3],[0.299, 0.587, 0.114])
    #This is where the gray scaling happens
    return matrix_image_grey, matrix_template_grey
    #The grey scaled image and template are returns



color_image_input = input('Name of color image file: ')
color_template_input = input('Name of color of template file: ')
```

```
#This is where the user inputs their image and template


colored_image = img.imread(color_image_input)

colored_template = img.imread(color_template_input)

#The files are then read as a matrix

img_matrix_grey, template_matrix_grey = grayscale(colored_image, colored_template)

#The grey scaled function from above is executed



type_of_algorithm = input('Enter which algorithm you want to use(SSD,CC,NCC,ALL): ')

#This is where the user types in which algorithm they want to use



if type_of_algorithm == 'SSD':

    start = time.time()

    ssd(colored_image,colored_template, img_matrix_grey,template_matrix_grey)

    end = time.time()

    print("--- %s seconds ---" % (end - start))
```

```
elif type_of_algorithm == 'CC':
    start = time.time()
    cc(colored_image, colored_template, img_matrix_grey,template_matrix_grey)
    end = time.time()
    print("--- %s seconds ---" % (end - start))

elif type_of_algorithm == 'NCC':
    start = time.time()
    ncc(colored_image,colored_template,img_matrix_grey,template_matrix_grey)
    end = time.time()
    print("--- %s seconds ---" % (end - start))

elif type_of_algorithm == 'ALL':
    start = time.time()

combined_algorithm(colored_image,colored_template,img_matrix_grey,template_matrix_grey)
    end = time.time()
    print("--- %s seconds ---" % (end - start))
```

#Depending on what the user types, one of the algorithms will be executed and the time

#each program takes is also printed

## Sum of Squared Differences Code:

```
import numpy as np

import matplotlib.pyplot as mpl

import matplotlib.patches as patches



def sumofsqdiff(img_matrix_grey,template_matrix_grey,w,h):



    W,H = img_matrix_grey.shape[0], img_matrix_grey.shape[1]



    S = np.zeros((W - w + 1, H - h + 1))



    #first sigma
```

```
    for k in range(0, W - w + 1):
        #second sigma
        for l in range(0, H - h + 1):
            #I is name of image
            #T is name of template
            S[k,l] = np.sum(np.square((img_matrix_grey[k:k+w,l:l+h] - template_matrix_grey[:,:])))
            #This finds the grey scaled S matrix

    reshaped_S= np.array(S)
    return reshaped_S.reshape((W-w)+1,(H-h)+1)

##########################################################################################

def maximum_value(S):
    R1 = np.array(S)
    minimum_S = np.where(R1 == np.min(S))
    return minimum_S[0],minimum_S[1]
    #This function finds the minimum entry in the S matrix.
```

```
#############################################################################
################

def output_ssd(colored_image,colored_template, img_matrix_grey,template_matrix_grey):
    h,w = colored_template.shape[0], colored_template.shape[1]
    S = sumofsqdiff(img_matrix_grey, template_matrix_grey,h,w)
    max_S_value = maximum_value(S)
    max_i,max_j = (max_S_value[0],max_S_value[1])
    #This part assigns little h and litte w to the dimensions of the template image
    #This part also exectues the two functions above
    #The output_ssd function will be executed in the main function


    mpl.imshow(S, cmap = 'gray')
    #This outputs the grey R_matrix output map for this alogrithm


    all_outputs = mpl.figure()
```

```
    all_outputs.add_subplot(2,1,1)

    current_axis1 = mpl.gca()

    current_axis1.imshow(colored_image)

    rect_img = patches.Rectangle((max_j,max_i), w, h, linewidth=1, edgecolor= 'g',
facecolor='none')

    current_axis1.add_patch(rect_img)

    #This creates a green rectangle around the first image



    i = int(max_i)

    j = int(max_j)

    y_initial = int(i-0.1*h)

    y_final = int(i + 1.1*h)

    x_initial = int(j - 0.1*w)

    x_final = int(j + 1.1*w)

    #These calculations make the size of the rectangle a little less than the size of the

    #cropped imag



    all_outputs.add_subplot(2,1,2)

    current_axis2 = mpl.gca()
```

```
    current_axis2.imshow(colored_image[y_initial : y_final , x_initial : x_final])

    rect_temp = patches.Rectangle((0.1*w ,0.1*h ), w, h, linewidth=3, edgecolor= 'g', facecolor='none')

    current_axis2.add_patch(rect_temp)

    #This creates a green rectangle around the cropped image
```

## Correlation Coefficient Code:

```
import numpy as np

import matplotlib.pyplot as mpl

import matplotlib.patches as patches


def corrcoef(mimag, mtemp,w,h):


    W, H = mimag.shape[0], mimag.shape[1]

    #W and H are the dimensions of the grey scaled image

    Tsum = np.sum(mtemp)

    Tresult = mtemp - (1/(w * h)) * Tsum

    Rsum = np.zeros((W-w+1, H-h+1))
```

```
    #Rsum is first a matrix of zeros
    for i in range (W - w):
        for j in range (H - h):
            Itemp = mimag[i:i+w,j:j+h]
            Isum = np.sum(Itemp)
            Iresult = Itemp - (1/(w * h)) * Isum


            Rsum[i,j] = np.sum(Iresult * Tresult)
    #This is where the calculation happens which gets the grey scaled R matrix


    reshaped_R= np.array(Rsum)
    return reshaped_R.reshape((W-w)+1,(H-h)+1)
    #This reshapes the R matrix into the dimensions of the image and returns it


################################################################################
##############################


def maximum_value(R):
    R1 = np.array(R)
```

```
    maximum_R = np.where(R1 == np.max(R))

    return maximum_R[0],maximum_R[1]

    #This function finds the maximum entry in the R_matrix and returns the i and j values

################################################################################
##############################


def output_cc(mimag, mtemp, img_matrix_grey,template_matrix_grey):

    h,w = mtemp.shape[0], mtemp.shape[1]

    R = corrcoef(img_matrix_grey, template_matrix_grey,h,w)

    max_R_value = maximum_value(R)

    max_i,max_j = (max_R_value[0],max_R_value[1])

    print(max_i,max_j)

    #This part assigns little h and litte w to the dimensions of the template image

    #This part also exectues the two functions above

    #The output_ncc function will be executed in the main function



    mpl.imshow(R, cmap = 'gray')

    #This outputs the grey scaled R matrix output map
```

```
    all_outputs = mpl.figure()



    all_outputs.add_subplot(2,1,1)
    current_axis1 = mpl.gca()
    current_axis1.imshow(mimag)
    rect_img = patches.Rectangle((max_j,max_i), w, h, linewidth=1, edgecolor= 'g',
facecolor='none')
    current_axis1.add_patch(rect_img)
    #This creates a green rectangle around the first image


    i = int(max_i)
    j = int(max_j)
    y_initial = int(i-0.1*h)
    y_final = int(i + 1.1*h)
    x_initial = int(j - 0.1*w)
    x_final = int(j + 1.1*w)
    #These calculations make the size of the rectangle a little less than the size of the
    #cropped image
```

```
    all_outputs.add_subplot(2,1,2)

    current_axis2 = mpl.gca()

    current_axis2.imshow(mimag[y_initial : y_final , x_initial : x_final])

    rect_temp = patches.Rectangle((0.1*w ,0.1*h ), w, h, linewidth=3, edgecolor= 'g', facecolor='none')

    current_axis2.add_patch(rect_temp)

    #This creates a green rectangle around the cropped image
```

## Normalized Cross-Correlation:

```
import matplotlib.pyplot as plt

import numpy as np

import matplotlib.patches as patches

#patches module is imported in order to draw the rectangles for the plots



def norm_cross_corr_calc(img_matrix_grey,template_matrix_grey,w,h):

    W, H = img_matrix_grey.shape[0], img_matrix_grey.shape[1]

    # W and H are set equal to the the dimensions of the grey scaled image
```

```
R_matrix = []


for i in range((W-w)+1):
    for j in range((H-h)+1):


        numerator_sum = np.sum(img_matrix_grey[i:i+w,j:j+h] * template_matrix_grey[:,:])
        first_denominator_sum = np.sum(np.sqrt((img_matrix_grey[i:i+w,j:j+h])**2))
        second_denominator_sum = np.sum(np.sqrt(template_matrix_grey[:,:]**2))
        #Since the Normalized Cross-Correlation has three summation to
        #it, each sum variable is a different part of the algorithm.
        #np.sum is used for each summation so that extra for loops aren't needed
        R_matrix.append(numerator_sum/(first_denominator_sum * second_denominator_sum))
        #Each R value calculated is appended into a list called R_matrix
reshaped_R= np.array(R_matrix)
return reshaped_R.reshape((W-w)+1,(H-h)+1)
#The list is converted into a matrix and then reshaped based on dimensions of the image
#and template
```

```
#################################################################################
################
def maximum_value(R):
    R1 = np.array(R)
    maximum_R = np.where(R1 == np.max(R))
    return maximum_R[0],maximum_R[1]
#This function finds the maximum entry in the R_matrix and returns the i and j values for
#that entry


#################################################################################
################
def output_ncc(colored_image,colored_template,img_matrix_grey,template_matrix_grey):
    h,w = colored_template.shape[0], colored_template.shape[1]
    R = norm_cross_corr_calc(img_matrix_grey, template_matrix_grey, h, w)
    max_R_value = maximum_value(R)
    max_i,max_j = (max_R_value[0],max_R_value[1])
    #This part assigns little h and litte w to the dimensions of the template image
    #This part also exectues the two functions above
    #The output_ncc function will be executed in the main function
```

```
    plt.imshow(R,cmap='gray')
    #This outputs the grey R_matrix output map for this alogrithm


    all_outputs = plt.figure()
    #Below a figure will be created of the image with where the object should be as well as
    #a crop of the image with where the object is
    all_outputs.add_subplot(2,1,1)
    current_axis1 = plt.gca()
    current_axis1.imshow(colored_image)
    rect_img = patches.Rectangle((max_j,max_i), w, h, linewidth=1, edgecolor= 'g',
facecolor='none')
    current_axis1.add_patch(rect_img)
    #This creates a green rectangle around the first image



    i = int(max_i)
    j = int(max_j)
    y_initial = int(i-0.1*h)
```

```
    y_final = int(i + 1.1*h)

    x_initial = int(j - 0.1*w)

    x_final = int(j + 1.1*w)

    #These calculations make the size of the rectangle a little less than the size of the

    #cropped image




    all_outputs.add_subplot(2,1,2)

    current_axis2 = plt.gca()

    current_axis2.imshow(colored_image[y_initial : y_final , x_initial : x_final])

    rect_temp = patches.Rectangle((0.1*w ,0.1*h ), w, h, linewidth=3, edgecolor= 'g',
facecolor='none')

    current_axis2.add_patch(rect_temp)

    #This creates a green rectangle around the cropped image
```

## Combined Algorithm Code:

```
import matplotlib.pyplot as plt

import numpy as np
```

```
import matplotlib.patches as patches

from PythonTeam15ProjectNormCCORR import norm_cross_corr_calc as ncc_calc

from PyTeam15ProjectSSD import sumofsqdiff as ssd_calc

from PyTeam15ProjectCCAlgorithm import corrcoef as cc_calc

#The functions that calculate the grey scaled for each algorithm are all imported above



def

combined_algorithm(colored_image,colored_template,img_matrix_grey,template_matrix_grey):

    h,w = colored_template.shape[0], colored_template.shape[1]

    R = ncc_calc(img_matrix_grey,template_matrix_grey,h,w)



    S = ssd_calc(img_matrix_grey,template_matrix_grey,h,w)



    R2 = cc_calc(img_matrix_grey, template_matrix_grey,h,w)

#Each function that creates the grey scaled image is executed



    ave_R_ncc = np.mean(R)

    std_R_ncc = np.std(R)

    new_R_ncc = (R - ave_R_ncc) / std_R_ncc
```

```
    ave_S_ssd = np.mean(S)

    std_S_ssd = np.std(S)

    new_S_ssd = ((1/S) - ave_S_ssd) / std_S_ssd

    #Since the ssd algorithm finds the minimun, the inverse of S is used so that we can find

    #the maximum


    ave_R2_cc = np.mean(R2)

    std_R2_cc = np.std(R2)

    new_R2_cc = (R2 - ave_R2_cc) / std_R2_cc


# In order to have all the matrices on the same scale, a mathematical equation is used. First,

# the average of the matrix is found. Then, the standard deviation of the matrix is found.

# Finally, to find the new, scaled matrix. You take the original image, subtract the average

# and divide all that by the standard deviation. You would do this for each of the three

# matrices
```

```
    combined_new_matrix = new_R_ncc + new_S_ssd + new_R2_cc


    max_R = np.where(combined_new_matrix == np.max(combined_new_matrix))

    max_i,max_j = max_R[0],max_R[1]


#The three matrices are all added and the max entry is found through np.max




    plt.imshow(combined_new_matrix,cmap='gray')

    #The greyscaled, scaled matrix is plotted


    all_outputs = plt.figure()




    all_outputs.add_subplot(2,1,1)

    current_axis1 = plt.gca()
```

```
    current_axis1.imshow(colored_image)

    rect_img = patches.Rectangle((max_j,max_i), w, h, linewidth=1, edgecolor= 'g', facecolor='none')

    current_axis1.add_patch(rect_img)

    #This creates a green rectangle around the first image

    i = int(max_i)

    j = int(max_j)

    y_initial = int(i-0.1*h)

    y_final = int(i + 1.1*h)

    x_initial = int(j - 0.1*w)

    x_final = int(j + 1.1*w)

    #These calculations make the size of the rectangle a little less than the size of the

    #cropped imag



    all_outputs.add_subplot(2,1,2)

    current_axis2 = plt.gca()

    current_axis2.imshow(colored_image[y_initial : y_final , x_initial : x_final])

    rect_temp = patches.Rectangle((0.1*w ,0.1*h ), w, h, linewidth=3, edgecolor= 'g', facecolor='none')
```

```
current_axis2.add_patch(rect_temp)

#This creates a green rectangle around the cropped image
```