The Report is Generated by DrillBit Plagiarism Detection Software

Submission Information

Author Name	ARNAV SINGH_A2305219087
Title	NTCC InHouse Report
Submission/Paper ID	318544
Submission Date	08-July-2021 10:05:37
Total Pages	16
Total Words	2645

Result Information

Similarity	5 %
Unique	95 %
Internet Sources	3 %
Journal/Publication Sources	1 %
Total content under 'Quotes'	5 %
Similarity content under 'Quotes'	5 %

Exclude Information

References/Bibliography	Excluded
Quotes	**Improper usage of 'Quotes'**
Sources: Less than 14 Words Similarity	Excluded

DrillBit
Anti Plagiarism Software

DrillBit Similarity Report

5	23	A
SIMILARITY %	MATCHED SOURCES	GRADE

A-Satisfactory (0-10%)
B-Upgrade (11-40%)
C-Poor (41-60%)
D-Unacceptable (61-100%)

Sl.No	LOCATION	MATCHED DOMAIN	%	SOURCE TYPE
1.	18	core.ac.uk	<1	Publication
2.	5	simplisafe.com	<1	Internet
3.	15	pythonbytes.fm	<1	Internet
4.	4	docs.opencv.org	<1	Internet
5.	13	express-rijopleiding.nl	<1	Internet
6.	12	drmccalley.com	<1	Internet
7.	14	www.itraintechnologies.com	<1	Internet
8.	16	Dayalbagh Educational Institute Student Thesis Published in inflibnet - www.inflibnet.ac.in	<1	Publication
9.	20	data-flair.training	<1	Internet
10.	22	www.int-arch-photogramm-remote-sens-spatial-inf-sci.net	<1	Publication

11.	17	My First Issue as Editor-in-Chief Notes from the Editor by Mohanty-2016	<1	Publication
12.	19	Principles of Secure Processor Architecture Design, by Szefer, Jakub- 2018	<1	Publication
13.	2	arxiv.org	<1	Publication
14.	23	Computer vision approach for phase identification from steel microstructure by Choudhury-2019	<1	Publication
15.	21	Review of Corporate Governance Practices and Financial Distress Prediction by Ahmad-2018	<1	Publication
16.	11	en.wikipedia.org	<1	Internet
17.	10	www.ijireeice.com	<1	Publication
18.	9	A Hybrid Vehicle Detection Method Basedon Viola-Jonesand H O G+ S V Mfrom U A V Images	<1	Journal-www.mdpi.com
19.	8	ro.uow.edu.au	<1	Publication
20.	7	biomedres.us	<1	Publication
21.	6	asp-eurasipjournals.springeropen.com	<1	Internet
22.	3	Infant Formulas for Food Allergy Treatment and Prevention by Parekh-2016	<1	Publication

23.	1	ROS-Based Human Detection and Tracking from a Wireless Controlled Mobile Robot U by Sankar-2019	<1	Publication

ABSTRACT

“Human detection and tracking are tasks of computer vision systems for locating and following people in video imagery. Human detection is the task of locating all instances of human beings present in an image, and it has been most widely accomplished by searching all locations in the image, at all possible scales, and comparing a small area at each location with known templates or patterns of people. Human tracking is the process of temporally associating the human detections within a video sequence to generate persistent paths, or trajectories, of the people. Human detection and tracking are generally considered the first two processes in a video surveillance pipeline and can feed into higher-level reasoning modules such as action recognition and dynamic scene analysis” [1].

INTRODUCTION

In this fast-advancing world, technology advancements wait for no one. By 2015, Tesla had already made its first self-driving auto-pilot car which could even achieve feats like drive itself to the destination even if the driver falls asleep. While keeping this advancement in our mind, we must dive inside in the lines of code that Tesla had to write to make a full functioning body.

In this project, we will only be paying attention to only one aspect of Tesla auto-pilot car which will be “Human Tracking and Detection”.

While it could be said that technology is being shrunk to such an extent that everything could be kept in our pockets, the security of such products should not be understated. The security of everything, be it our products itself, phones, wallets or even our homes for this case, should be the number one priority.

Let us take an example of a billion-dollar company known as SimpliSafe. SimpliSafe is a home security system which has an expert human-tracking algorithm which tells the user of the product on its smartphone-friendly application about who arrives at your house, who is at the door and a lot of tweaks here and there. This simple yet effective security system and its algorithm is something that can be achieved through machine learning.

In this project, we would focus on simplifying this human identifying system and try to replicate some functions that the SimpliSafe program uses.

RELATED WORK

“The constructive need for robots to coexist with humans requires human–machine interaction. It is a challenge to operate these robots in such dynamic environments, which requires continuous decision-making and environment-attribute update in real-time. An autonomous robot guide is well suitable in places such as museums, libraries, schools, hospital, etc” [3].

“Detecting human beings accurately in a visual surveillance system is crucial for diverse application areas including abnormal event detection, human gait characterization, congestion analysis, person identification, gender classification and fall detection for elderly people. The first step of the detection process is to detect an object which is in motion. Object detection could be performed using background subtraction, optical flow, and spatial-temporal filtering techniques. Once detected, a moving object could be classified as a human being using shape-based, texture-based, or motion-based features. A comprehensive review with comparisons on available techniques for detecting human beings in surveillance videos is presented in this paper. The characteristics of few benchmark datasets as well as the future research directions on human detection have also been discussed” [4].

“With crimes on the rise all around the world, video surveillance is becoming more important day by day. Due to the lack of human resources to monitor this increasing number of cameras manually new computer vision algorithms to perform lower and higher-level tasks are being developed. We have developed a new method incorporating the most acclaimed Histograms of Oriented Gradients the theory of Visual Saliency and the saliency prediction model Deep Multi Level Network to detect human beings in video sequences” [5].

The above research paper excerpts suggest the urging need of Human Detection systems to be introduced and be advanced enough to work together with the technology boom. Since security concerns have been rising exponentially, it is of paramount importance that we develop a good and functional system that will help us in doing this.

LITERATURE REVIEW

1.1 HISTOGRAM OF ORIENTED GRADIENTS (HOG)

"The histogram of oriented gradients (HOG) is a feature descriptor used in computer vision and image processing for the purpose of object detection. The technique counts occurrences of gradient orientation in localized portions of an image" [2].

"The essential thought behind the histogram of oriented gradients descriptor is that local object appearance and shape within an image can be described by the distribution of intensity gradients or edge directions. The image is divided into small, connected regions called cells, and for the pixels within each cell, a histogram of gradient directions is compiled. The descriptor is the concatenation of these histograms. For improved accuracy, the local histograms can be contrast-normalized by calculating a measure of the intensity across a larger region of the image, called a block, and then using this value to normalize all cells within the block. This normalization results in better invariance to changes in illumination and shadowing" [2].

1.2 SUPPORT VECTOR MACHINE (SVM)

"In machine learning, support-vector machines are supervised learning models with associated learning algorithms that analyse data for classification and regression analysis" [2].

For our convenience, OpenCV has already been implemented in an efficient way to combine the HOG Descriptor algorithm with Support Vector Machine or SVM.

1.3 PYTHON

"Python is an interpreted high-level general-purpose programming language. Python's design philosophy emphasizes code readability with its notable use of significant indentation. Its language constructs as well as its object-oriented approach aim to help programmers write clear, logical code for small and large-scale projects."

“Python is dynamically-typed and garbage-collected. It supports multiple programming paradigms including structured, object-oriented and functional programming. Python is often described as a "batteries included" language due to its comprehensive standard library.”

“Guido van Rossum began working on Python in the late 1980s, as a successor to the ABC programming language, and first released it in 1991 as Python 0.9.0. Python 2.0 was released in 2000 and introduced new features, such as list comprehensions and a garbage col lection system using reference counting. Python 3.0 was released in 2008 and was a major revision of the language that is not completely backward-compatible and much Python 2 code does not run unmodified on Python 3.” [6]

2.1 PROJECT REQUISITES

In our project, we will be requiring these Python libraries for the functioning and implementation of our program:

- OPENCV: “OpenCV is the huge open-source library for the computer vision, machine learning, and image processing and now it plays a major role in real-time operation which is very important in today's systems. By using it, one can process images and videos to identify objects, faces, or even handwriting of a human.” [6]
- IMUTILS: Imutils are a series of convenience functions to make basic image processing functions such as translation, rotation, resizing, skeletonization, and displaying images.
- NUMPY: NumPy is a Python library used for working with arrays. It also has functions for working in domain of linear algebra, Fourier transform, and matrices.
- ARGPARSE: Argparse is the “recommended command-line parsing module in the Python standard library.” It is what you use to get command line arguments into your program.

```
1  import cv2
2  import imutils
3  import numpy as np
4  import argparse
```

2.1.1 OPEN CV

"OpenCV was started at Intel in 1999 by **Gary Bradsky** , and the first release came out in 2000. **Vadim Pisarevsky** joined Gary Bradsky to manage Intel's Russian software OpenCV team. In 2005, OpenCV was used on Stanley, the vehicle that won the 2005 DARPA Grand Challenge. Later, its active development continued under the support of Willow Garage with Gary Bradsky and Vadim Pisarevsky leading the project. OpenCV now supports a multitude of algorithms related to Computer Vision and Machine Learning and is expanding day by day." [6]

"OpenCV supports a wide variety of programming languages such as C++, Python, Java, etc., and is available on different platforms including Windows, Linux, OS X, Android, and iOS. Interfaces for high-speed GPU operations based on CUDA and OpenCL are also under active development." [6]

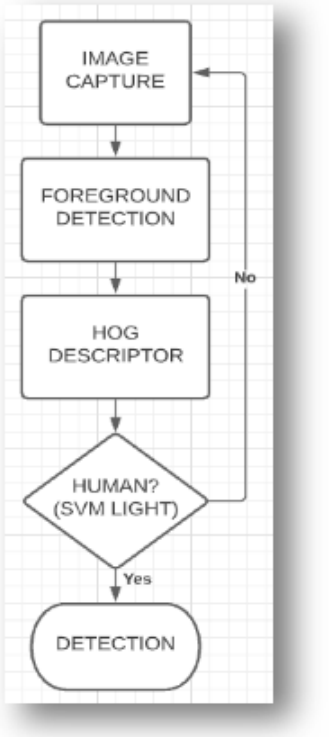
"OpenCV-Python is the Python API for OpenCV, combining the best qualities of the OpenCV C++ API and the Python language." [6]

The project is wholly based on this library. All the various commands and implementations that have been used have been made possible due to this library.

METHODOLOGY

3.1 CREATING MODEL FOR HUMAN DETECTION

Now, we shall use the HOG Descriptor which is readily available for us in OpenCV. The functioning will be as the flowchart below suggests:



We shall start by an image capture which could be in 3 ways – which would be suggested later in the report. The program then detects everything that the capture has to offer.

From there, our HOG Descriptor will be implemented. By working together with the SVM, it will detect whether the lighting on the capture resembles a human or not.

If yes, the program will output that a human has been detected or else, it will reroute to the start of the image capture or end the program if the user demands it.

3.2 BRAIN OF THE CODE

The actual functionality of the code will start from here. The "**detect( )**" method will come in handy for us. **Detect()** will take the image capture; in this case, a video would be taken as an example; it shall take the video and divide it in multiple individual frames and will continue to detect humans frame by frame and, return the output frame after frame so that it presumably appears as if a video is playing.

While returning the output, it will put green boxes around each human which it detects in the frame.

The "**detectMultiScale()**" will help us in making the bounding box for human detection. It works based on co-ordinates, which are in the form of X, Y, W, H.

X and Y shall be the starting coordinates of the box and W and H would be the width and height of the box, respectively.

```
6   def detect(frame):
7       bounding_box_cordinates, weights =  HOGCV.detectMultiScale(frame, winStride = (4, 4), padding = (8, 8), scale = 1.03)
8
9       person = 1
10      for x,y,w,h in bounding_box_cordinates:
11          cv2.rectangle(frame, (x,y), (x+w,y+h), (0,255,0), 2)
12          cv2.putText(frame, f'person {person}', (x,y), cv2.FONT_HERSHEY_COMPLEX_SMALL, 0.5, (0,0,255), 1)
13          person += 1
14
15      cv2.putText(frame, 'Status : Detecting ', (40,40), cv2.FONT_HERSHEY_COMPLEX_SMALL, 0.8, (255,0,0), 2)
16      cv2.putText(frame, f'Total Persons : {person-1}', (40,70), cv2.FONT_HERSHEY_COMPLEX_SMALL, 0.8, (255,0,0), 2)
17      cv2.imshow('output', frame)
18
19      return frame
```

3.3 EYES OF THE CODE (DETECTOR)

We shall now commence to work on inputs. We have decided to keep the inputs in structure of a live webcam feed, a video format (mp4, wmv, avi, mov etc) and a still image format (jpg, png, gif, eps etc).

We make a method now namely **humanDetector()** which will take an argument that could be any of the three options that we listed earlier.

If the argument tells us that the input would be a live feed, then the camera argument will be turned to true or else, the camera would not be used.

A writer variable is also added for us to act as a flag for the camera. Writer is set to “None” as default but if the camera is active, writer is given some attributes for to save the video feed that the camera displays.

If and Elif statements have also been added for the respective arguments.

```
81  def humanDetector(args):
82      image_path = args["image"]
83      video_path = args['video']
84      if str(args["camera"]) == 'true' : camera = True
85      else : camera = False
86
87      writer = None
88      if args['output'] is not None and image_path is None:
89          writer = cv2.VideoWriter(args['output'],cv2.VideoWriter_fourcc(*'MJPG'), 10, (600,600))
```

3.4 DETECTION BY CAMERA

The piece de resistance of our entire project is to have a live feed and have an overlay of human detection over it. To achieve this, we will need to us a method in the OpenCV requisite which will be **VideoCapture()**.

"OpenCV allows a straightforward interface to capture live stream with the camera (webcam). It converts video into grayscale and display it."

"We need to create a **VideoCapture** object to capture a video. It accepts either the device index or the name of a video file. A number which is specifying to the camera is called device index. We can select the camera by passing the O or 1 as an argument. After that we can capture the video frame-by-frame."

For each frame, we will call the **detect()** method and simultaneously write the frame in our output file.

```
50  def detectByCamera(writer):
51      video = cv2.VideoCapture(0)
52      print('Detecting people...')
53
54      while True:
55          check, frame = video.read()
56
57          frame = detect(frame)
58          if writer is not None:
59              writer.write(frame)
60
61          key = cv2.waitKey(1)
62          if key == ord('q'):
63                  break
64
65      video.release()
66      cv2.destroyAllWindows()
```

3.5 DETECTION BY VIDEO AND IMAGE

Our inputs will also support images and videos. For that we have used a similar method to that of our detection by camera one. Except in this one, our input will have a provided path that we will also input.

Several conditions will have to be added as well for error checking here. The path that we would input into the terminal could be incorrect or non-existent.

Just like the previous method, the implementation will be the same. The major difference would be that we will check whether the program checks and reads the frame properly. When the video ends (i.e., the frame could not be read) we terminate said loop.

```
def detectByPathVideo(path, writer):


    video = cv2.VideoCapture(path)
    check, frame = video.read()
    if check == False:
        print('Video Not Found. Please Enter a Valid Path (Full path of Video Should be Provided).')
        return

    print('Detecting people...')
    while video.isOpened():

        check, frame =  video.read()

        if check:
            frame = imutils.resize(frame , width=min(800,frame.shape[1]))
            frame = detect(frame)

            if writer is not None:
                writer.write(frame)

            key = cv2.waitKey(1)
            if key == ord('q'):
                break
        else:
            break
    video.release()
    cv2.destroyAllWindows()
```

```
def detectByPathImage(path, output_path):
    image = cv2.imread(path)

    image = imutils.resize(image, width = min(800, image.shape[1]))

    result_image = detect(image)

    if output_path is not None:
        cv2.imwrite(output_path, result_image)

    cv2.waitKey(0)
    cv2.destroyAllWindows()
```

IMPLEMENTATION

4.1 USER INTERFACE

We have mainly used Argparse as an interface that could be used to input objects and commands through the command line itself making the program very easy to execute and device friendly.

4.1.1 ARGPARSE

The Argparse module makes it easy to write user-friendly command-line interfaces. The program defines what arguments it requires, and Argparse will figure out how to parse those out of sys.argv.

The Argparse module also automatically generates help and usage messages and issues errors when users give the program invalid arguments. The command-line interface is also known as the CLI, which interacts with a command-line script. Python provides many libraries that allow us to work with the CLI, but Python Argparse is the most suitable library in the current scenario.

There will be Three arguments within the Parser:

1. **Image:** The path to the image file inside your system
2. **Video:** The path to the Video file inside your system
3. **Camera:** A variable that if set to 'true' will call the cameraDetect() method.

```
101 def argsParser():
102     arg_parse = argparse.ArgumentParser()
103     arg_parse.add_argument("-v", "--video", default=None, help="path to Video File ")
104     arg_parse.add_argument("-i", "--image", default=None, help="path to Image File ")
105     arg_parse.add_argument("-c", "--camera", default=False, help="Set true if you want to use the camera.")
106     arg_parse.add_argument("-o", "--output", type=str, help="path to optional output video file")
107     args = vars(arg_parse.parse_args())
108 
109     return args
```

4.2 PRESENTING OUTPUT

Since we have 3 ways to output, we shall go one by one understanding what each output does-

For scanning a video file, we would type in:

TERMINAL: python [PYTHON FILE NAME] -v '[PATH]'

```
PS C:\Users\Arnav\Desktop\NTCC\code> python main.py -v 'vid.mp4'
```

For scanning an image:
TERMINAL: python [PYTHON FILE NAME] -i '[PATH]'

```
PS C:\Users\Arnav\Desktop\NTCC\code> python main.py -i 'pic.jpg'
```

To use the camera:
TERMINAL: python [PYTHON FILE NAME] -c true

```
PS C:\Users\Arnav\Desktop\NTCC\code> python main.py -c true
```

To save the output of the camera output:
TERMINAL: python [PYTHON FILE NAME] -c true -o '[FILE_NAME]'

```
PS C:\Users\Arnav\Desktop\NTCC\code> python main.py -c true -o 'output1.mp4'
```

EXPERIMETATION RESULTS

5.1 OBSERVATIONS

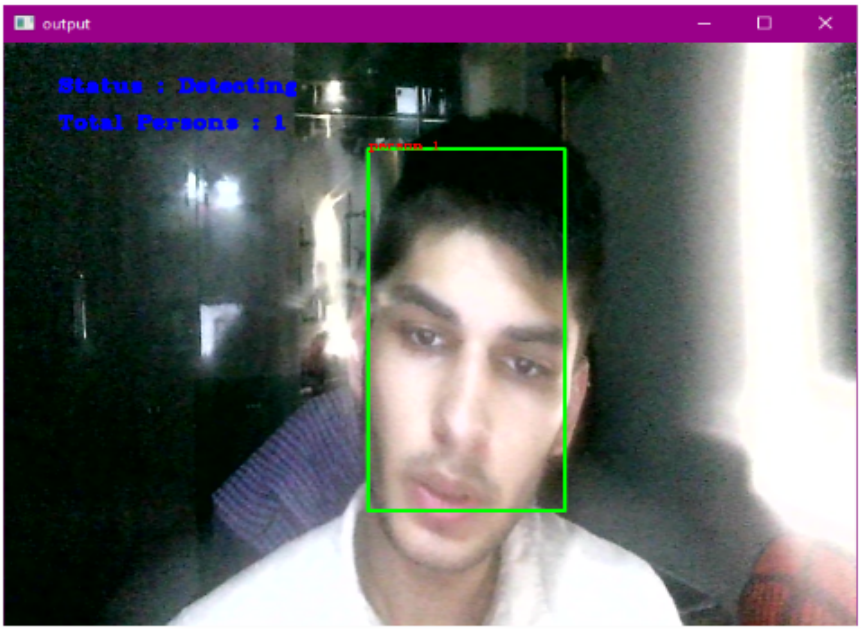
The program is successfully able to detect humans by using its functions of HOG. It can distinguish between different lightings of background and therefore is able to mark down human silhouettes. HOG decomposes an image into small, squared cells, computes a histogram of oriented gradients in each cell, normalizes the result using a block-wise pattern, and return a descriptor for each cell.

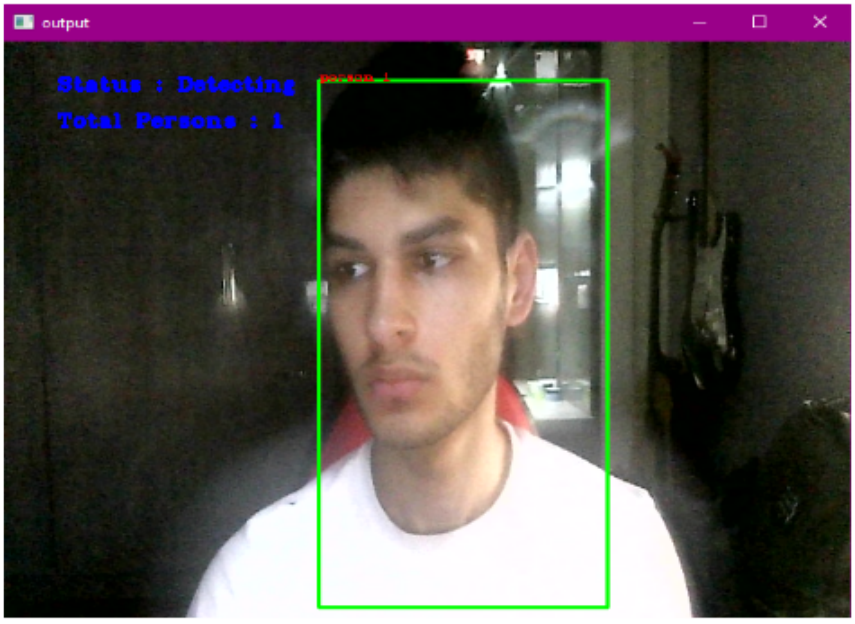
5.2 RESULTS

The results are quite promising. Every method was able to detect a person successfully.
When we input a video, it was able to separately distinguish between different persons.

Same occurred when we input an image, it was successful in tracking a person.

And when we opened the camera, it recognized the subject in front of it and was able to successfully track it.





CONCLUSION AND FUTURE SCOPE

The project mainly focuses on recognizing and tracking a person in a video. The program takes a video or a sequence of videos as input and outputs the names of the people who are seen in the videos. The assumption is that the videos are shot in a homogeneous environment. The program needs some processing time so for an input video of 30 frames per second the output comes out to be a video of 10-15 fps i.e., the program is system dependent. The program tries to avoid the false positives as much as possible but for certain textures it still becomes confused. The program successfully detects the presence of a person in a video and is also capable in detecting the faces of these people in a video and if the program succeeds in recognizing the person the name of the person is outputted on the terminal thus helping the user to track any person given a sequence of videos.

In the future this project could be improved by improving the performance of this code. It could also be beneficial to improve the human recognition of the code.