EE 417 TERM PROJECT

FALL 2021

Accident Detection in Traffic

Berk Açıkgöz – 26631

Yusufhan Kırçova – 26678

Problem Definition and Objective

Each year, 1.35 million people are killed on roadways, and every day almost 3,700 people are killed globally in motor vehicle crashes (WHO,2021). It is estimated to be the 8th leading cause of death globally and leading cause of death for children and young adults 5-29 years of age. It is estimated that crash injuries cost the world approximately 1.8 trillion dollars. Statistics show that low- and middle-income countries are more affected by motor vehicle crashes. Crash death rates are over three times higher in low-income countries. While possessing %60 of the world's registered vehicles, low-income countries are responsible for the %90 of the world's motor vehicle deaths.

Project objective is to develop a software, which can detect motor vehicle crashes as soon as it occurs, depending on parameters such as vehicle speed and vehicle direction. Furthermore, a system in which the emergency contacts of the drivers and local emergency services are notified of the crash, can be used to provide help to the injured people. This project is related to Feature detection, Object Recognition, Visual Tracking Motion Estimation and Anomaly Detection. A system in which can

Next steps for our project would be to predict the car crashes beforehand using live footage and warning the drivers. Therefore, the system might prevent accidents before they occur.

Problem Formulation and Solution Method

Given a video input, in order to detect accidents in traffic we must utilize Object detection, Object tracking to gather vehicle movement data. Using the data gathered from the input, unordinary movement will be analyzed using speed, acceleration anomaly, change in angle anomaly and vehicle overlap. Our analysis will provide an integer which will show if an accident occurs or not in the given footage.

The accident detection algorithm proposed by Ijjina et al (2019) consists of two main parts:

- Vehicle Detection and Vehicle Tracking
- Accident Detection

A. Vehicle Detection and Vehicle Tracking

First part of the project was to detect the vehicles in a given image using object detection frameworks. This was necessary since the project requires objects to be tracked in order to gather vehicle movement data. YOLOv3 (You Only Look Once, Version 3) is a real-time object detection algorithm that identifies specific objects in videos, live feeds, or images (Redmon et al, 2016). YOLOv3 was used to detect vehicles in a given image. Centroid tracking was utilized to achieve object tracking, so that moving objects can be identified and movement data such as speed, acceleration and movement trajectory can be extracted (Nascimento et al, 1999).

B. Accident Detection

We implemented an accident detection algorithm which was used to detect car crashes using anomalies in vehicle movement. This algorithm and along with some threshold information have been published by IEEE and have been used in our project.

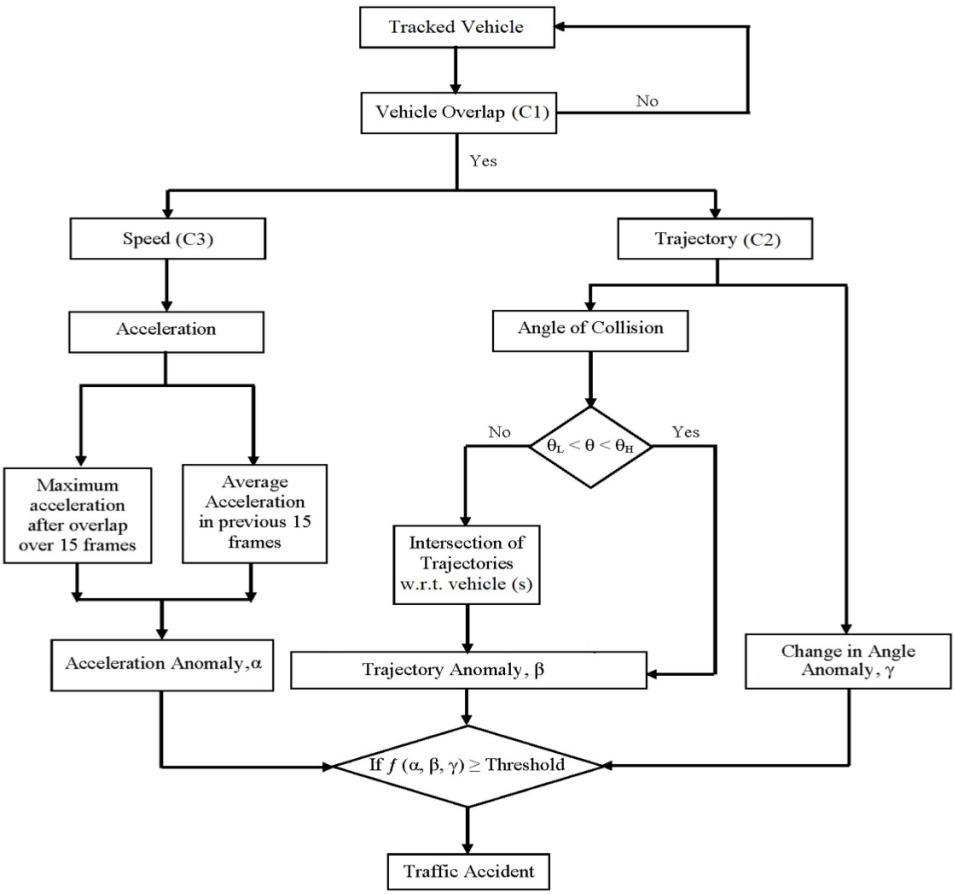

fig. 1. Workflow of the algorithm

First condition for an algorithm to consider a possibility of crash is the position of each vehicle. For a vehicle to be crashed with another, their respective positions must overlap. This condition is checked using the positions we have found with

Object Tracking. The equation for this condition is given below. If the condition below holds up then this will indicate that an overlap occurs.

$$(2\times|a.x-b.x| < a.\alpha+b.\alpha) \wedge (2\times|a.y-b.y| < a.\beta+b.\beta)$$

Second part of the tree divides into two sub-parts, with each part calculating a different measure for the probability of an accident.

C2) Trajectory

In order to determine the trajectories of the vehicles we use the centroids of vehicles for a time interval. Taking the differences between five successive centroids will give us a vector in which the trajectories of the vehicle will be inclined to move.

$$\text{magnitude} = \sqrt{(\boldsymbol{\mu}.i)^2 + (\boldsymbol{\mu}.j)^2}$$

We are required to determine the angle between different trajectories by using the angle between two vector formulas which can be seen below.

$$\theta = \arccos\left(\frac{\boldsymbol{\mu_1} \cdot \boldsymbol{\mu_2}}{|\boldsymbol{\mu_1}||\boldsymbol{\mu_2}|}\right)$$

C3) Speed

In order to determine the speed of the vehicles we estimate τ, the interval between the frames of the video, using the Frames Per Second (FPS).

$$\tau = \frac{1}{\text{FPS}}$$

During our implementation we used a constant FPS of 60 Frames per second. The distance covered by each vehicle is found by getting the centroids of vehicles over five frames. Gross Speed (Sg) is determined by the centroid difference (c1,c2) over five frames of time. The interval is the parameter which is determined by the number of frames which are taken into consideration while calculating the centroids, in this case we have fixed it to five frames, since it gave us more accurate results.

$$S_g = \frac{c_2 - c_1}{\tau \times Interval}$$

Then gross speed is normalized by the irrespective of its distance from the camera. Normalization was calculated by taking video frame (H), height of bounding box (h), width of the video frame(W), width of the bounding box and Gross Speed (Sg). Scaled Speeds are calculated by normalizing the gross speed.

$$S_s = \left(\frac{H - h}{H} + 1\right) \times S_g$$

Since we will be taking into consideration both acceleration and speed, we must calculate the Acceleration of the vehicle for a given interval. Since, acceleration is

the difference in speed between an interval of time such calculation can be made by the following formula below.

$$A = \frac{S_s^2 - S_s^1}{\tau \times Interval}$$

Using all of the variables which have been collected by different operations we are going to calculate the probability of an accident. Acceleration Anomaly, Trajectory Anomaly and Change in Angle Anomaly are all taken into consideration while calculating the final value. The combination of all the parameters are used in a function which takes into weightages of each individual threshold. At last if the final value is greater than 0.5 then it is considered as a vehicular accident.

Implementation and Results

Jupyter-notebook was used during the implementation since it helped us to run python scripts part by part without having to go over the whole code block.

For the first step of the implementation, we developed a structure in which a particular mp4 file is divided into frame images which can be used in later stages of the code. Since we were frequently testing if the outputs are correct or not, this part was essential to the project and prioritized among other parts.

```
import cv2
vidcap = cv2.VideoCapture('videoplayback.mp4')
success,image = vidcap.read()
count = 0
while success:
  cv2.imwrite("frame%d.jpg" % count, image)
  success,image = vidcap.read()
  count += 1
```

OpenCV was utilized and a while loop is used to go over each frame of the image.

Object Detection

We applied the YOLOv3 Algorithm to detect objects in given frames (Redmon et al, 2016) (Garima13a, 2018). We have used opencv to load images, and a modified version of darknet which is a deep neural network written by the creators of YOLO Algorithm. This specific version of the darknet was modified in order to work with PyTorch. However, since we are not going to be doing any neural network training, we are going to use a pre-trained set of weights which were initially trained on Common Objects in Context. The weights can be found under the Issues tab in the referred GitHub project, please see References. Since the CPU was being utilized by Darknet each individual image took around 5-10 seconds to complete its detection. We set nms threshold to 0.6 and iou threshold to 0.4.

```
# Set the IOU threshold. Default value is 0.4
iou_thresh = 0.4
# Set the NMS threshold. Default value is 0.6
nms_thresh = 0.6
# Detect objects in the image
boxes = detect_objects(m, resized_image, iou_thresh, nms_thresh)
# Print the objects found and the confidence level
print_objects(boxes, class_names)
#Plot the image with bounding boxes and corresponding object class labels
plot_boxes(original_image, boxes, class_names, plot_labels = True)
```

YOLOv3 provides results with high accuracy and high confidence. Most of the images we tested had almost perfect results. Since we were only looking at cars, trucks and busses; YOLOv3 was more than enough for our project. The detected objects were held in a box array which will be used throughout the implementation of other steps. Below images are some examples of object detection using YOLOv3

Figure 2: Object Detection using YOLOv3

Figure 3: Object Detection using YOLOv3

Object Tracking

The centroid tracking algorithm was used to track the objects detected with YOLO v3 (Nascimento et al, 1999). The algorithm works on the simple principle that a centroid (center of the rectangle of the detected object) in one frame must be closest to the centroid that belongs to the same object in the next frame, even if the object moves. An implementation of centroid tracking was utilized alongside YOLO v3 to track the moving vehicles (Rosebrock, 2018). Some snapshots that display the tracked objects can be found below.

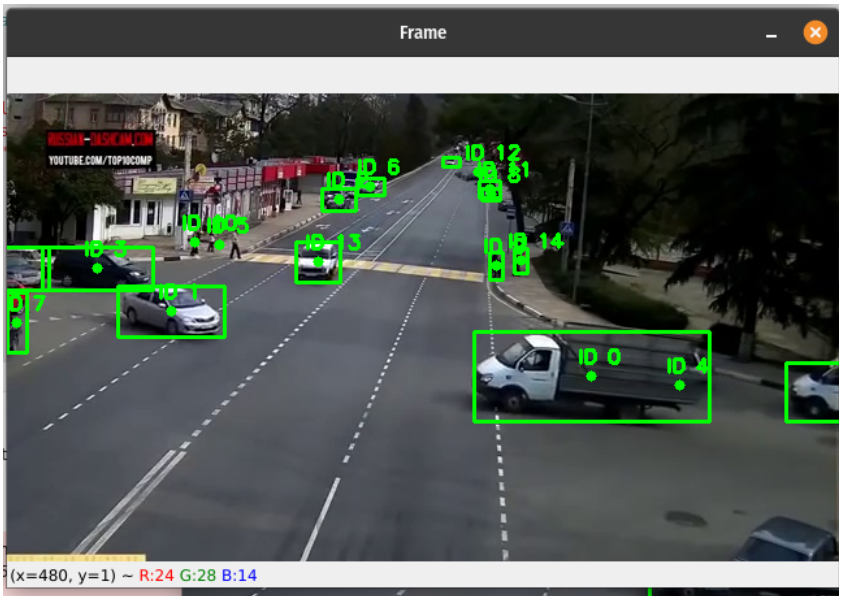

Fig. 4.1: The state of tracked objects just before an accident.

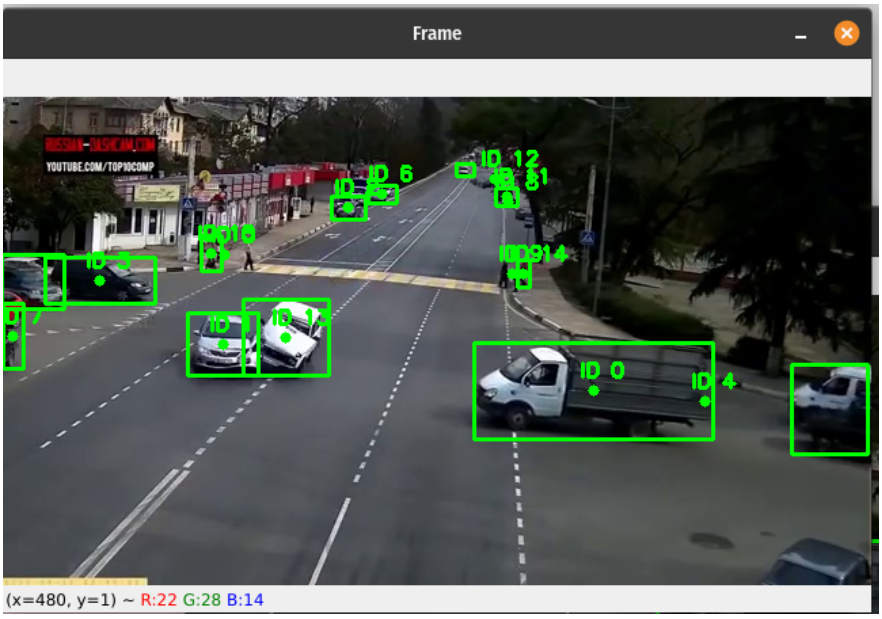

Figure 4.2: The tracked objects at the moment of the accident

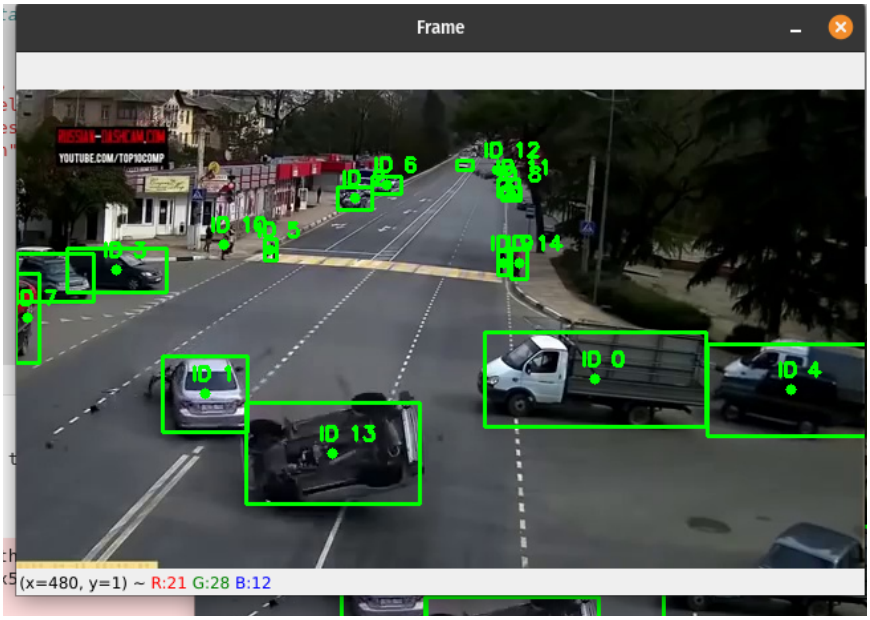

Figure 4.3: The tracked objects at the after the accident

Accident Detection

The angle and acceleration at given moments were calculated as per explained in the Problem Formulation and Solution Method section. The three parameters; Acceleration Anomaly (α), Trajectory Anomaly (β) and Change in Angle Anomaly (ɣ) were calculated in light of the methodology explained in (Ijjina et al., 2019).

For the acceleration anomaly, the difference between the maximum acceleration in the 15 frames following the collision and the average acceleration in the 15 frames preceding the accident was calculated. The maximum value for any given object pair for the entire video was taken as premises to decide on the value of alpha. For the mapping function, please refer to the Appendix, under the "acceleration anomaly detection" section.

For the trajectory anomaly, $-\pi/4$ was selected as θ_{low} and $\pi/4$ was selected as θ_{high} and beta was chosen as either 0 or 0.1, (refer to Appendix). For the angles outside that interval, their absolute value was taken and they were scaled with $\pi/2$, which became the value of beta.

For the change in angle anomaly, the difference between the angles in two adjacent frames were taken. For any vehicle pair, the maximum of these angle differences were taken as premises for the value of gamma. The maximum angle difference was mapped to certain values of gamma, which increase faster as the angle increases and saturates at gamma = 1 when the value of the angle becomes 0.8 radians. For the entire mapping function, please refer to the Appendix, under the “change in angle anomaly detection” section.

Finally, these three parameters are all given weights, chosen as 0.2 for alpha, 0.35 for beta and 0.45 by gamma in our case, and if the weighted sum of these three parameters exceed 0.5, it is decided that an accident occured between the given vehicle/object pair.

Discussion of Results

In our first trial, there were only two accidents detected: between object ID 0 and 4, and between object ID 1 and 13. However, only the accident between ID 1 and 13 actually occurred, which has a larger score than the other detected accident. The output is given below:

Final score for pair (0, 4) is 0.775180899378008

Final score for pair (1, 13) is 0.8751938894351401

The false detection for object pair (0, 4) is due to object 0 and 4 obstructing each other for the bigger part of the video. This obstruction results in very similar features to the case of an actual accident, hence the false positive. A better video angle can prevent such cases being detected as false positives.

Below are the results from a different trial:

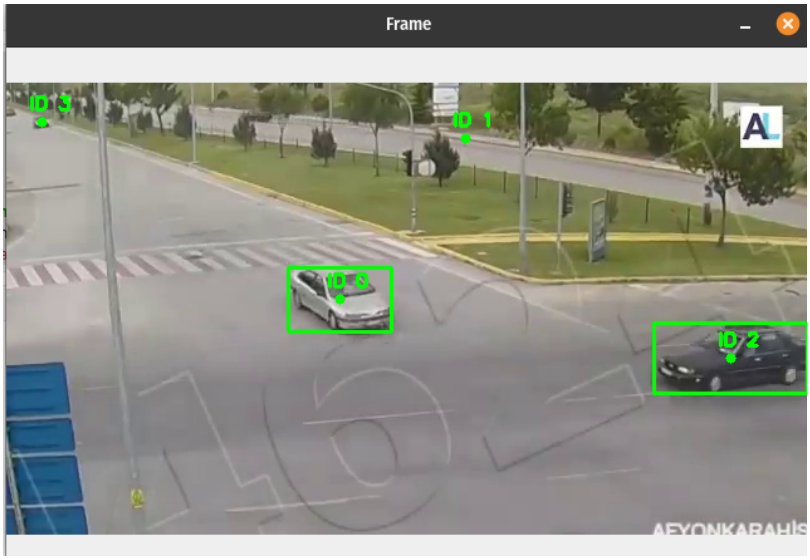

Figure 5.1 The tracked objects at the before the accident

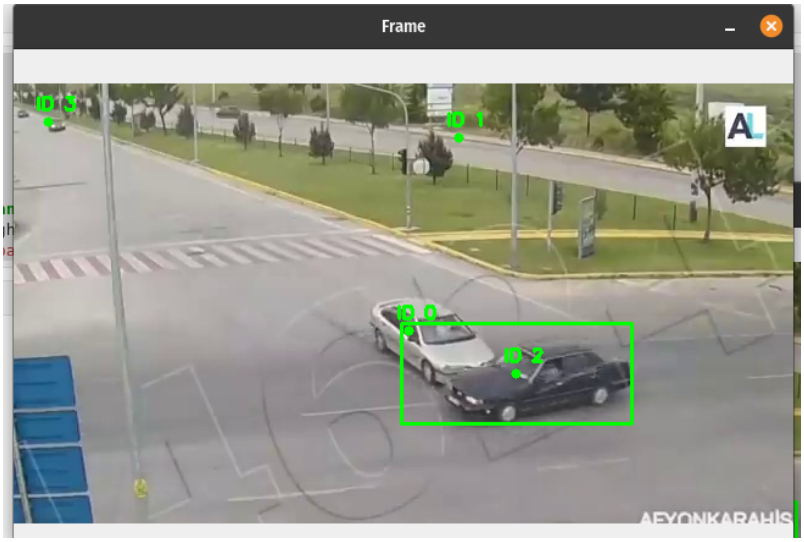

Figure 5.2 The tracked objects at the time of the accident

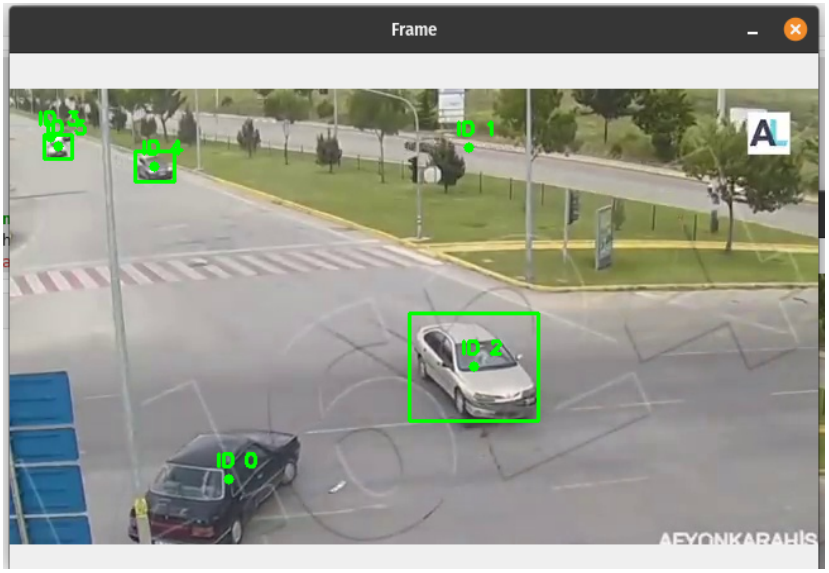

Figure 5.3 The tracked objects at the after the accident

The final results for the video are as follows:

Final score for pair (0, 2) is 0.6652992124147674

Final score for pair (3, 5) is 0.08

The detection between objects 0 and 2 are successfully detected, whereas the interaction between objects 3 and 5 are not deemed as an accident, as the total score is below 0.5. As no vehicle is obstructed for a long time by another, no such false positives are observed.

Resources

Garima13a (2018). YOLO Object Detection. Retrieved from https://github.com/Garima13a/YOLO-Object-Detection on January 8, 2022.

Ijjina, E. P., Chand, D., Gupta, S., & Goutham, K. (2019). Computer vision-based accident detection in traffic surveillance. *2019 10th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*. https://doi.org/10.1109/icccnt45670.2019.8944469

Nascimento, J. C., Abrantes, A. J., & Marques, J. S. (1999). An algorithm for centroid-based tracking of moving objects. *1999 IEEE International Conference on Acoustics, Speech, and Signal Processing. Proceedings. ICASSP99 (Cat. No.99CH36258)*. https://doi.org/10.1109/icassp.1999.757548

Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You only look once: Unified, real-time object detection. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. https://doi.org/10.1109/cvpr.2016.91

Redmon Joseph (2013–2016). Darknet: Open Source Neural Networks in C http://pjreddie.com/darknet/

Redmon Joseph (2018). An Incremental Improvement https://pjreddie.com/darknet/yolo/

Rosebrock, A. (2018). Simple object tracking with OpenCV. Retrieved from https://www.pyimagesearch.com/2018/07/23/simple-object-tracking-with-opencv/ on December 30, 2021.

World Health Organization. (2021). *Road traffic injuries*. World Health Organization. Retrieved January 15, 2022, from https://www.who.int/news-room/fact-sheets/detail/road-traffic-injuries

Appendix A: Main Program

```
# import the necessary packages
from tracker.centroidtracker import CentroidTracker
from imutils.video import VideoStream
import numpy as np
import argparse
import imutils
import time
import cv2
import youtube_dl
from math import sqrt, acos, pi

import cv2
import matplotlib.pyplot as plt
```

```
from utils import *
from darknet import Darknet

# Set the location and name of the cfg file
cfg_file = './cfg/yolov3.cfg'

# Set the location and name of the pre-trained weights file
weight_file = './weights/yolov3.weights'

# Set the location and name of the COCO object classes file
namesfile = 'data/coco.names'

# Load the network architecture
m = Darknet(cfg_file)

# Load the pre-trained weights
m.load_weights(weight_file)

# Load the COCO object classes
class_names = load_class_names(namesfile)

# Set the NMS threshold
nms_thresh = 0.6
```

```
# Set the IOU threshold
iou_thresh = 0.4

# initialize our centroid tracker and frame dimensions
ct = CentroidTracker()

# dictionary to keep centroid values in
centroids_objects = {}

accelerations = {}
norm_differences = {}
prev_scaled_speeds = {}
overlaps = {}
angles = {}

start = 3700
fps = 10.0
frame_inc = 5
T = frame_inc/fps
idx = start

# loop over the frames from the video stream
while True:
```

```
        idx += frame_inc

        frame = cv2.imread('frames/frame%d.jpg' % idx)

        iterated_objects = []

        # Set the default figure size
        plt.rcParams['figure.figsize'] = [24.0, 14.0]

        # Convert the image to RGB
        original_image = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)

        # We resize the image to the input width and height of the first layer of the network.
        resized_image = cv2.resize(original_image, (m.width, m.height))

        # Detect objects in the image
        detections = detect_objects(m, resized_image, iou_thresh, nms_thresh)

        # Print the objects found and the confidence level
        #print_objects(detections, class_names)

        width = frame.shape[1]
        height = frame.shape[0]
```

```
boxes = []
confidences = []
classIDs = []

for detection in detections:
confidence = float(detection[5])
classID = int(detection[6])
x1 = int(np.around((detection[0] - detection[2]/2.0) * width))
y1 = int(np.around((detection[1] - detection[3]/2.0) * height))
x2 = int(np.around((detection[0] + detection[2]/2.0) * width))
y2 = int(np.around((detection[1] + detection[3]/2.0) * height))
if confidence > 0.5:
box = [x1, y1, x2, y2]
boxes.append(box)
confidences.append(confidence)
classIDs.append(classID)
#print(box)
cv2.rectangle(frame, (x1, y1), (x2, y2), (0, 255, 0), 2)
cv2.imshow('window', frame)

# update our centroid tracker using the computed set of bounding
# box rectangles
objects = ct.update(boxes)
```

```
#print(objects)


# loop over the tracked objects

for (objectID, centroid) in objects.items():

# draw both the ID of the object and the centroid of the

# object on the output frame

text = "ID {}".format(objectID)


# if the object ID is already registered, append the newest centroid value at the end of the list

if objectID in centroids_objects:

centroids_objects[objectID].append(centroid)

# if the centroid is new, register it in the dictionary

else:

centroids_objects[objectID] = [centroid]


cv2.putText(frame, text, (centroid[0] - 10, centroid[1] - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 2)

cv2.circle(frame, (centroid[0], centroid[1]), 4, (0, 255, 0), -1)


interval = 5

thres_diff = 15.0 # for magnitude of differences


for item, cents in centroids_objects.items():
```

```
if len(cents) > interval and item in objects: # if there are at least 'interval' centroid values recorded and the object has not been deregistered from the tracker

# difference between two points

diff = cents[-1] - cents[-1-interval]

#print(diff)

mag_diff = (diff[0]**2 + diff[1]**2)**0.5

#print(mag_diff)

norm_diff = (diff[0]/float(mag_diff), diff[1]/float(mag_diff)) if mag_diff != 0 else (0.0, 0.0)


if mag_diff > thres_diff:

    if item not in norm_differences:

    norm_differences[item] = [[norm_diff, mag_diff, idx]]

    else:

    norm_differences[item].append([norm_diff, mag_diff, idx])


# speed and acceleration

gross_speed = (cents[-1] - cents[-1-interval])/(T*interval)

scaled_speed = (gross_speed[0]*((width-ct.widthheight[item][0])/width + 1), gross_speed[1]*((height-ct.widthheight[item][1])/height + 1))

if item in prev_scaled_speeds: # if a previous scaled speed value exists

    acc = ((scaled_speed[0]**2 - prev_scaled_speeds[item][0]**2)/(T*interval), (scaled_speed[1]**2 - prev_scaled_speeds[item][1]**2)/(T*interval))

    if item not in accelerations:
```

```
                accelerations[item] = [[acc, idx]]
                else:
                accelerations[item].append([acc, idx])
        prev_scaled_speeds[item] = scaled_speed

        for (object1, centroid1) in centroids_objects.items():
        if object1 not in objects:
        continue
        iterated_objects.append(object1)
        c1 = centroid1[-1]
        wh1 = ct.widthheight[object1]
        for (object2, centroid2) in centroids_objects.items():
        if object2 not in objects:
                continue
        if object2 not in iterated_objects:
                c2 = centroid2[-1]
                wh2 = ct.widthheight[object2]
                if (2*abs(c1[0] - c2[0]) < (wh1[0] + wh2[0])) and (2*abs(c1[1] - c2[1])
< (wh1[1] + wh2[1])):
                pair = (object1, object2)
                # record the moment of overlap and the overlapping object pair
                if pair in overlaps:
                overlaps[pair].append(idx)
                else:
```

```
                overlaps[pair] = [idx]


                if object1 in norm_differences and object2 in norm_differences:
                # find angle between two overlapping objects
                ratio = norm_differences[object1][-1][0][0] * norm_differences[object2][-1][0][0] + norm_differences[object1][-1][0][1] * norm_differences[object2][-1][0][1]
                if ratio > 1.0:
                        ratio = 1.0
                elif ratio < -1.0:
                        ratio = -1.0
                theta = acos(ratio)
                if theta >= pi/2:
                        theta = theta - pi
                # record the angle between two objects and the time
                if pair in angles:
                        angles[pair].append([theta, idx])
                else:
                        angles[pair] = [[theta, idx]]
                else:
                theta = None
                #print("Theta for overlap of " + str(object1) + " and " + str(object2) + " is " + str(theta))


        # show the output frame
```

```
        cv2.imshow("Frame", frame)
        key = cv2.waitKey(1) & 0xFF



# do a bit of cleanup
cv2.destroyAllWindows()


# acceleration anomaly detection


# amounts to go back and forward to from the point of contact
go_back_frame_no = 15
go_forward_frame_no = 15


accanomalies = {}


for pair, idxs in overlaps.items():
        # for each overlapping moment of a pair
        alphamax = 0
        for frameIdx in idxs:
        # check if there are at least 15 acceleration values before the overlapping condition for both objects
        obj1accbefore = []
        obj2accbefore = []
```

```
obj1accafter = []

obj2accafter = []


for items in accelerations[pair[0]]:

obj1accbefore += [items] if items[1] < frameIdx else [] # add acceleration values to list if it is before the collision

obj1accafter += [items] if items[1] > frameIdx else [] # add acceleration values to list if it is after the collision


for items in accelerations[pair[1]]:

obj2accbefore += [items] if items[1] < frameIdx else [] # add acceleration values to list if it is before the collision

obj2accafter += [items] if items[1] > frameIdx else [] # add acceleration values to list if it is after the collision


if len(obj1accbefore) >= go_back_frame_no and len(obj2accbefore) >= go_back_frame_no and len(obj1accafter) >= go_forward_frame_no and len(obj2accafter) >= go_forward_frame_no:

# calculate the magnitude of the acceleration and add to the average

obj1beforeavg = 0.0

for acc in obj1accbefore[-go_back_frame_no:]:

    obj1beforeavg += (acc[0][0]**2 + acc[0][1]**2)**0.5

obj1beforeavg /= float(go_back_frame_no)


obj2beforeavg = 0.0

for acc in obj2accbefore[-go_back_frame_no:]:
```

```
        obj2beforeavg += (acc[0][0]**2 + acc[0][1]**2)**0.5
obj2beforeavg /= float(go_back_frame_no)

# calculate the magnitude of the acceleration and find maximum
obj1aftermax = 0.0
for acc in obj1accafter[0:go_forward_frame_no]:
        magn = (acc[0][0]**2 + acc[0][1]**2)**0.5
        obj1aftermax = magn if magn > obj1aftermax else obj1aftermax

obj2aftermax = 0.0
for acc in obj2accafter[0:go_forward_frame_no]:
        magn = (acc[0][0]**2 + acc[0][1]**2)**0.5
        obj2aftermax = magn if magn > obj2aftermax else obj2aftermax

obj1accdiff = obj1aftermax - obj1beforeavg
obj2accdiff = obj2aftermax - obj2beforeavg

acc_anomaly_score = obj1accdiff + obj2accdiff

# determine alpha
if abs(acc_anomaly_score) < 100:
        alpha = 0
elif abs(acc_anomaly_score) < 500:
        alpha = 0.2
```

```
        elif abs(acc_anomaly_score) < 750:
                alpha = 0.4
        elif abs(acc_anomaly_score) < 900:
                alpha = 0.6
        elif abs(acc_anomaly_score) < 1200:
                alpha = 0.8
        elif abs(acc_anomaly_score) < 1500:
                alpha = 0.9
        else:
                alpha = 1

        alphamax = alpha if alpha > alphamax else alphamax

        #print(pair, frameIdx, obj1accdiff, obj2accdiff)

        accanomalies[pair] = alphamax

print(accanomalies)

# trajectory anomaly detection

trajecanomalies = {}

theta_low = -pi/4
```

```
theta_high = pi/4

for pair, items in angles.items():
        betamax = 0
        for item in items:
        # if theta is between theta_low and theta_high
        if theta_low < item[0] < theta_high:
        if abs(item[0]) < (abs(theta_low) + abs(theta_high))/4.0:
                beta = 0.0
        else:
                beta = 0.1
        else:
        beta = abs(item[0])/(pi/2)

        betamax = beta if beta > betamax else betamax

        trajecanomalies[pair] = betamax

print(trajecanomalies)

# change in angle anomaly detection

changeanomalies = {}
```

```
for pair, items in angles.items():
        angle_diffs = []
        max_diff = 0
        for i in range(1, len(items)):
        diff = items[i][0] - items[i-1][0]
        if diff > pi/2:
        diff -= pi
        if abs(diff) > max_diff:
        max_diff = abs(diff)

        # decide gamma
        if max_diff < 0.1:
        gamma = 0
        elif max_diff < 0.2:
        gamma = 0.1
        elif max_diff < 0.3:
        gamma = 0.2
        elif max_diff < 0.4:
        gamma = 0.3
        elif max_diff < 0.5:
        gamma = 0.5
        elif max_diff < 0.6:
        gamma = 0.7
        elif max_diff < 0.7:
```

```
        gamma = 0.8
        elif max_diff < 0.8:
        gamma = 0.9
        else:
        gamma = 1

        changeanomalies[pair] = gamma

print(changeanomalies)

# decide the final outcome

alpha_weight = 0.2
beta_weight = 0.35
gamma_weight = 0.45

for pair in accanomalies:
        if pair in trajecanomalies and pair in changeanomalies:
        final_score = alpha_weight*accanomalies[pair] + beta_weight*trajecanomalies[pair] + gamma_weight*changeanomalies[pair]
        print("Final score for pair", pair, "is ", final_score)
```

Appendix B: Program to Convert mp4 File to Frames

```
import cv2
vidcap = cv2.VideoCapture('videoplayback.mp4')
success,image = vidcap.read()
count = 0
while success:
  cv2.imwrite("frame%d.jpg" % count, image)          # save frame as JPEG file
  success,image = vidcap.read()
  #print('Read a new frame: ', success)
  count += 1
print("DONE!")
```