DOCK MANAGEMENT SYSTEM USING OBJECT DETECTION MECHANISM

Documentation submitted to INDICOLD as per requirement of Internship

Submitted By

TEAM BROWN

- **A.B.Kamalesh (TL)**
- **B. Bhavyasree**
- **Darshan Kumar**
- **Neeluru Bhavana**
- **Priyanka J**

Under the guidance of
Mr. Kartik Jalan
COO of Kailash Agro cold Storage

INDICOLD INTERNSHIP

KAILASH AGRO PRIVATE LIMITED

PALWAL DISTRICT, HARIYANA – 121102

MAY 2021

ABSTRACT

Cold storage technologies provide services to extend the useful life time of short lived, decomposable items by providing thermal insulation to keep temperature sensitive objects. However, this emerging industry need solutions for various complications arising at each step from storage warehousing to product delivery. Considering the problem at docking at warehouse, there is a high possibility of accidents due to movement of trucks and other transportation vehicles. Our project aims to provide a solution to this problem using in trend object detection mechanisms. The main focus of the safety issue includes, ensuring safety while vehicle is being reversed and detection of obstruction in the path of the vehicle and dock. Mechanisms are also developed to determine whether a vehicle is entering or leaving the dock by determining the direction with appropriate notification setting.

TABLE OF CONTENTS

CHAPTER	NAME	PAGE NO.
1	INTRODUCTION	1
2	RELATED WORKS	2
3	RUDIMENTARY KNOWLEDGE	3
4	PROPOSED METHODOLOGY	7
5	CODING	14
6	RESULT AND ANALYSIS	27
7	CONCLUSION	30
8	BIBLIOGRAPHY	31

LIST OF FIGURES

FIGURE NO.	FIGURE NAME
1	Anchor Box Example
2	IoU Example
3	CNN Architecture Example
4	YOLOv4 tiny
5	OpenCV
6	Flowchart for Object Detection Algorithm
7	Flowchart for Object Tracking Algorithm
8	Alert Message Output
9	Closeness Of truck detected by Algorithm
10	Direction Detection By Object Tracking Algorithm
11	Camera Angle Dependency

CHAPTER 1
INTRODUCTION

Warehouse safety is important any industry as it goes conjointly with production. Even though safety standards to be followed and implied at a workhouse place improve but still there are many complications that are needed to be taken care of. One of the most serious type of accidents are caused due to vehicles in warehouse like trucks for transporting goods or forklifts. Although these vehicles are necessity and reduce the workload on a large scale but this may lead to carelessness. These accidents can be of two types, either driving into something or mishandling material. Considering the worse case scenario in which a person is hit would cause serious problem.

However, the recent trends of automation have led to reducing human effort and errors precisely. Evolution of computer vision and object recognition has led to boost in automation applications in the field of industry, transportation and robotics. This project is a step towards ensuring safety for workers at dock of cold storage through object detection mechanism which is the recent trend. The project proposes the use of object detection algorithm like YOLO and combines it with various computer vision functionalities provided by OpenCV library. We detect the vehicles with major focus on trucks and measure a safe distance to be kept for any object present in its path, including human. In the end we also provide pros and cons for the model proposed and its technical specification before industry deployment.

CHAPTER 2
RELATED WORKS

Over the past few years, a lot of research has been done on improving inventory management, supply chain management and application of new technologies in warehouse management systems. AI has played a key role in this evolutionary research. However, the application of AI has produced some unique application in robotics like components pickup from various locations at warehouse or algorithms for customer demand prediction and order fulfilment as in [1], [2], [3], [4], [6]. Bandaru et al [6] used genetic algorithm and devised an unsupervised learning algorithm for automatic identification and material handling for application inventory management. Similarly, Dou et al [3], used genetic algorithms to solve scheduling problems for warehouse management. Although many of the works were based on combining warehouse management and AI but still safety complications are needed to be taken into account. There are many applications that account for security and business domains using AI but very less has been done on object identification.

Object detection is a sub domain of computer vision. There are two different approaches to apply object detection into use. One is to use OpenCV which is an open-source computer vision application library and the other one is to use classical model for the same. OpenCV provides a lot of APIs for computer vision which include contour detection, colour tracking, space tracking, cascade classifiers for various purposes for classification, tracking and counting objects. Similarly in classical models, frameworks are built for particular application or various algorithms are tried and the optimal solution is found for the problem. Most widely used algorithms in this context are the feature extraction models. Gong et al [7] has analysed a series of approaches to extract features for classification problems. In our project we use YOLO v4 with OpenCV to use object detection for detecting trucks and vehicles and also try to track them to predict their direction.

CHAPTER 3
RUDIMENTARY KNOWLEDGE

❖ DATASET

Before using any image classification algorithm on a real-time application, these algorithms are required to be trained on large amount of input data. The dataset used for training any image detection algorithm uses input image, from which the features are detected by the algorithm and label, which classifies the image. The labels in image classification datasets consists of the name of the object and hence the algorithm determines that a particular set of features defines this object. After the image classification algorithm is trained, next step is to train the model for object detection and tracking. For this the dataset is modified for having image as input and bounding boxes as labels.

❖ BOUNDING BOXES AND ANCHOR BOXES

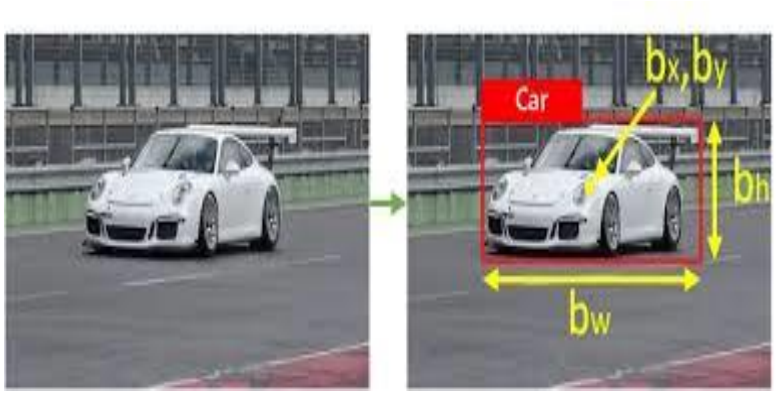

Fig1. Anchor box example

Whenever an image is encountered, it is resized to a particular shape and divided into a coordinate plane with bottom right corner as the origin. So, the bounding boxes used, simply put are the ‘x’ and ‘y’ coordinates with corresponding width ‘w’ and height ‘h’ of the box. The algorithm tries to learn what object is detected and how to localise the object by providing a box around it.

But objects of similar shape may confuse the notion of the learned feature of the algorithm. For example, an apple and a pear appear of similar sizes or a car image side view and front view maybe different and hence the size of bounding box. So, to distinguish these biases and classify the object to correct label, we associate a number of boxes (of the format (x, y, w, h)) to each label.

❖ INTERSECTION OVER UNION (IoU)

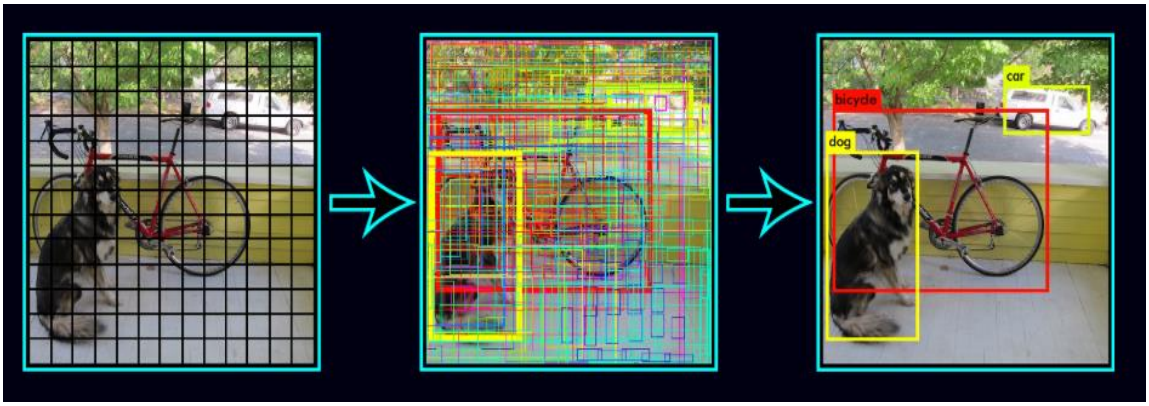

Fig2. IoU Example

Whenever an object detector or tracker tries to bound an image in using a box there are many prediction boxes, creating a mesh of boxes. The anchor box will decide which bounding box will have highest overlap by non-overlap. This concept is called Intersection over Union (IoU). A confidence measure is set for higher and lower limits of IoU. If the detected box overlap with the anchor box is more than the upper limit of threshold then the detection is valid. The label associated with the highest overlap anchor box is the class label for the detected object.

❖ CONVOLUTION NEURAL NETWORK

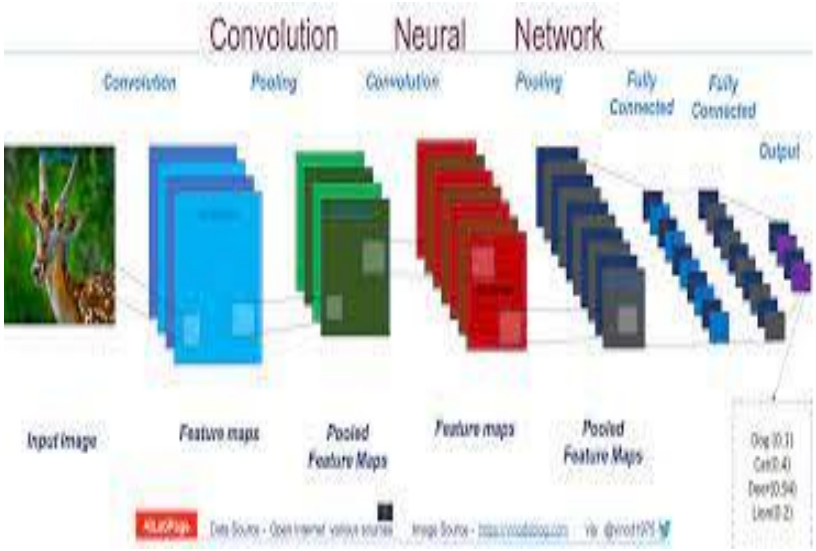

Fig3. CNN Architecture Example

Convolution Neural Networks (CNN) are the basic building blocks of many state-of-the-art image classification, object detection, and object tracking models. As the name suggests, each layer of the neural network performs a convolution operation to detect various features like heights, widths, contours etc. Any classifier requires higher input network to detect multiple small sized objects, more layers to capture increased size of input, and more parameters to detect variety of objects in a given image. In this project 'YOLO v4' is the model used for object detection and tracking.

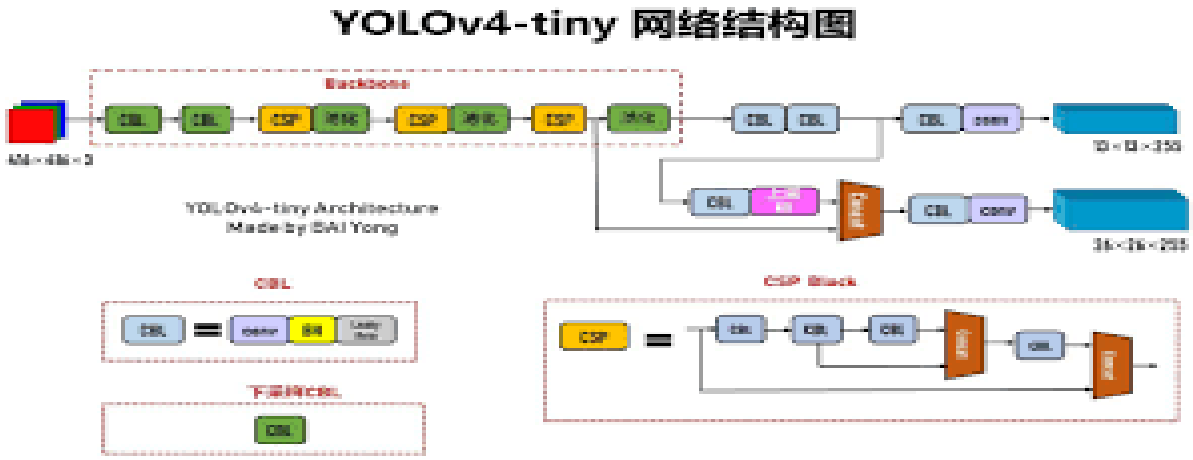

Fig4. YOLOv4 tiny

Some salient features of YOLO v4 are:

- The backbone of the model is CSPDarknet53 which consists of 29 3 × 3 convolution layers, 725 × 725 receptive field and 27.6 million parameters. So, the backbone detects variety of objects due to complex this complex architecture
- The neck of the architecture consists of a SPP (Spatial Pyramid Pooling) block and PANet (Path Aggregation Network). SPP increases the receptive field of the architecture to gather more features while the PANet is used to aggregate parameters from different backbones.
- The head of the architecture uses YOLO v3 and additional improvements have been made for efficient and faster training like Self-Adversarial training, optimal hyper-parameters for genetic algorithms training etc.

❖ OPENCV

Fig5. OpenCV

OpenCV is an open-source Computer Vision library which provides various tools for real-time computer vision applications. In this project, the library is used for reading input stream of videos, image preprocessing, providing input to the designated algorithm for predictions, drawing bounding box enclosing the object in the image and converting the frames back to output video.

CHAPTER 4
PROPOSED METHODOLOGY

The working of the algorithm is divided into two parts, object detection and object tracking. Object detection algorithms are mainly used for scanning objects and categorizing them in a video or an image. Similarly, object tracking algorithms are used for detecting an object (either specific or all objects) and track them in consecutive frames.

In this project, the concept of transfer learning is used. Transfer learning refers to using pre-trained models as a starting point instead of coding and training models from scratch. In this project pre-trained weights of YOLO-v4 tiny are used. The model was trained on COCO dataset for image classification and PASCAL VOC dataset for object detection. The configuration and weights files are out 20 MB which means the weights are condensed for faster detection. The speed is detected on an average of 0.22 fps.

The platform used for implementation is Google Colab with python 3.7.

- **OBJECT DETECTION**

  As mentioned, the main focus of the object detection is to detect the truck and the obstruction in the path. Therefore, the workflow of the project is as shown in the flowchart (fig. 1) below.

  1. The configuration files and weights files stored in the google drive are imported. The necessary versions of the packages are downloaded. Since Google Colab is used as platform, only OpenCV version 3.4 was installed.
  2. All the necessary packages are then imported and helper functions for Truck detection, distance computation and alert messaging were defined.
  3. Labels, weight and configuration files are loaded and the “readNetFromdarknet()” function from OpenCV library initialises the network for prediction using the weights and configuration.
  4. All the unconnected outputs from layers of the network are identified.
  5. Using OpenCV, video stream is initialised for reading the frames from video and a pointer to output file is determined. After that an approximate estimation of completion is provided by determining the total number of frames.
  6. A loop is created to read all the frames in the videos stream. For each frame grabbed, we perform image preprocessing via “blobfromImage()” function. Using this function, we resize the input image to the spatial size required by the CNN,

change the image channels to RGB format from the default input of BGR format, and provide a scale factor for scaling the image after mean subtraction.

7. We give the preprocessed output blob as input to our model and make detections. We initialise the lists for detected bounding boxes, confidences and class labels. We would also use a list to store mid-points or centroids of bounding boxes for detected objects.
8. Then, we loop over our detections and append the different value measures for each prediction to the lists initialised if the detection has surpassed the given confidence threshold for the particular class label.
9. In the next step, we apply Non-Maxima Suppression to the bounding boxes using confidence score provided to the function "NMSBoxes()".
10. We initialise the parameter called Pixel-Per-Metric ratio, for removing orientation defects from image by scaling the distance in real time scenario to the one in image frame captured by the camera. This means to calculate how many pixels in an image will measure a distance of 1 metre in real world. A distance of 6 ft is used as the distance metric to determine if the object is close enough for accident.
11. Here the user defined helper function, "TruckDetected()" is used to determine if a truck is detected. If the flag value corresponds to True then we use find out indexes of detections which are in close proximity to the truck, which may be any object detected (other vehicle or any obstruction). For this second helper function "compute_distance()" is used.
12. If there are objects in proximity less than the set threshold value then the object is bounded as a red box and an alert message is sent.
13. The alert message is sent using a third-party API Fast2SMS. An account was created in the website and using the authorization key provided a helper function "SendSMS()" is written in python to post a request to send request to the list of mobile numbers.
14. Then a writer is initialised using the OpenCV library to write the frames with bounding boxes printed on them and stored in an output file.
15. After all the frames in video are looped over, the writer and video stream reader are released.

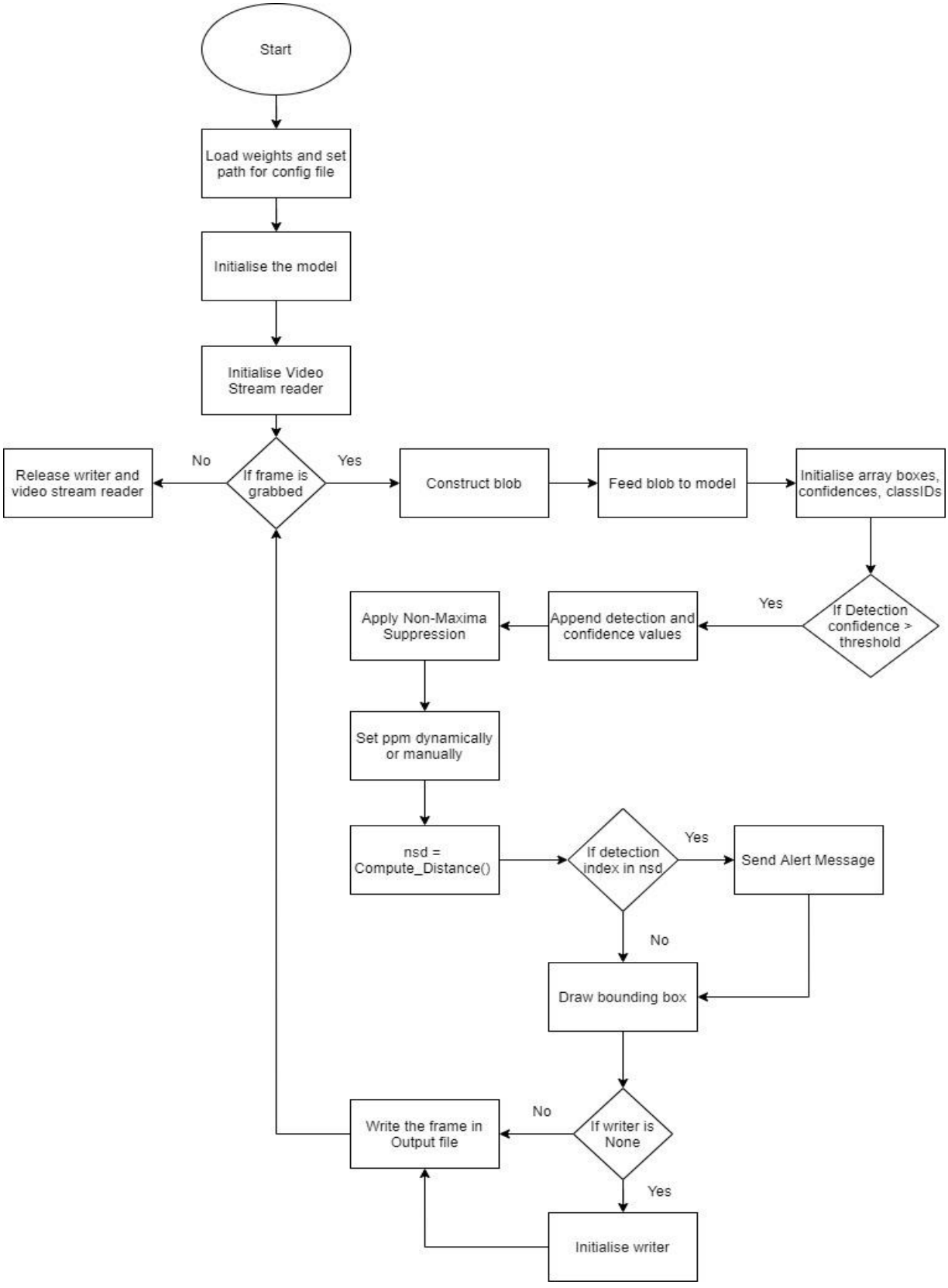

Fig6. Flowchart for Object Detection Algorithm

➢ OBJECT TRACKING

The main focus of object tracking is to detect the direction of movement of vehicle in and out of the dock. For the same, a new class “Centroid Tracker” is devised which registers each object detected with a unique ID. After detection from each frame, the centroid tracker tries to match the detections with the existing IDs by measuring the distance between the centroids of the detected object with the already detected ones. The idea is, in each frame the centroid of the same object would have moved the least distance. For each detection object the centroids are queued to some maximum buffer threshold provided. After the buffer is full, the direction of the object is determined on the basis of (x, y) coordinates of the centroid. Similarly, a threshold is kept for deregistering the object not present in frame. If the object is missing for a given number of frames, then the object ID is deregistered. The workflow is as shown in fig. 2.

1. The configuration files and weights files stored in the google drive are imported. The necessary versions of the packages are downloaded. Since Google Colab is used as platform, only OpenCV version 3.4 was installed.
2. “CentroidTracker” class blueprint is created with the required functions to register an object, deregister an object, track the object and detect the direction of the object.
3. All the necessary packages are then imported and helper functions for Truck detection, distance computation and alert messaging were defined.
4. Labels, weight and configuration files are loaded and the “readNetFromdarknet()” function from OpenCV library initialises the network for prediction using the weights and configuration.
5. All the unconnected outputs from layers of the network are identified.
6. Using OpenCV, video stream is initialised for reading the frames from video and a pointer to output file is determined. After that an approximate estimation of completion is provided by determining the total number of frames.
7. An object for centroid tracker class is initialised and a loop is created to read all the frames in the videos stream. For each frame grabbed, we perform image preprocessing via “blobfromImage()” function. Using this function, we resize the input image to the spatial size required by the CNN, change the image channels to RGB format from the default input of BGR format, and provide a scale factor for scaling the image after mean subtraction.

8. We give the preprocessed output blob as input to our model and make detections. We initialise the lists for detected bounding boxes, confidences and class labels. We would also use a list to store mid-points or centroids of bounding boxes for detected objects.
9. Then, we loop over our detections and append the different value measures for each prediction to the lists initialised if the detection has surpassed the given confidence threshold for the particular class label.
10. In the next step, we apply Non-Maxima Suppression to the bounding boxes using confidence score provided to the function "NMSBoxes()".
11. After all the detections are obtained, bounding boxes are created around the objects detected in that frame and then these boxes are given to the centroid tracker class object to update the tracked objects. If a new object is found, then it is registered to the class object.
12. After the detections are updated, the direction of each tracked object is obtained and written as text in the frame.
13. Then a writer is initialised using the OpenCV library to write the frames with bounding boxes printed on them and stored in an output file.
14. After all the frames in video are looped over, the writer and video stream reader are released.

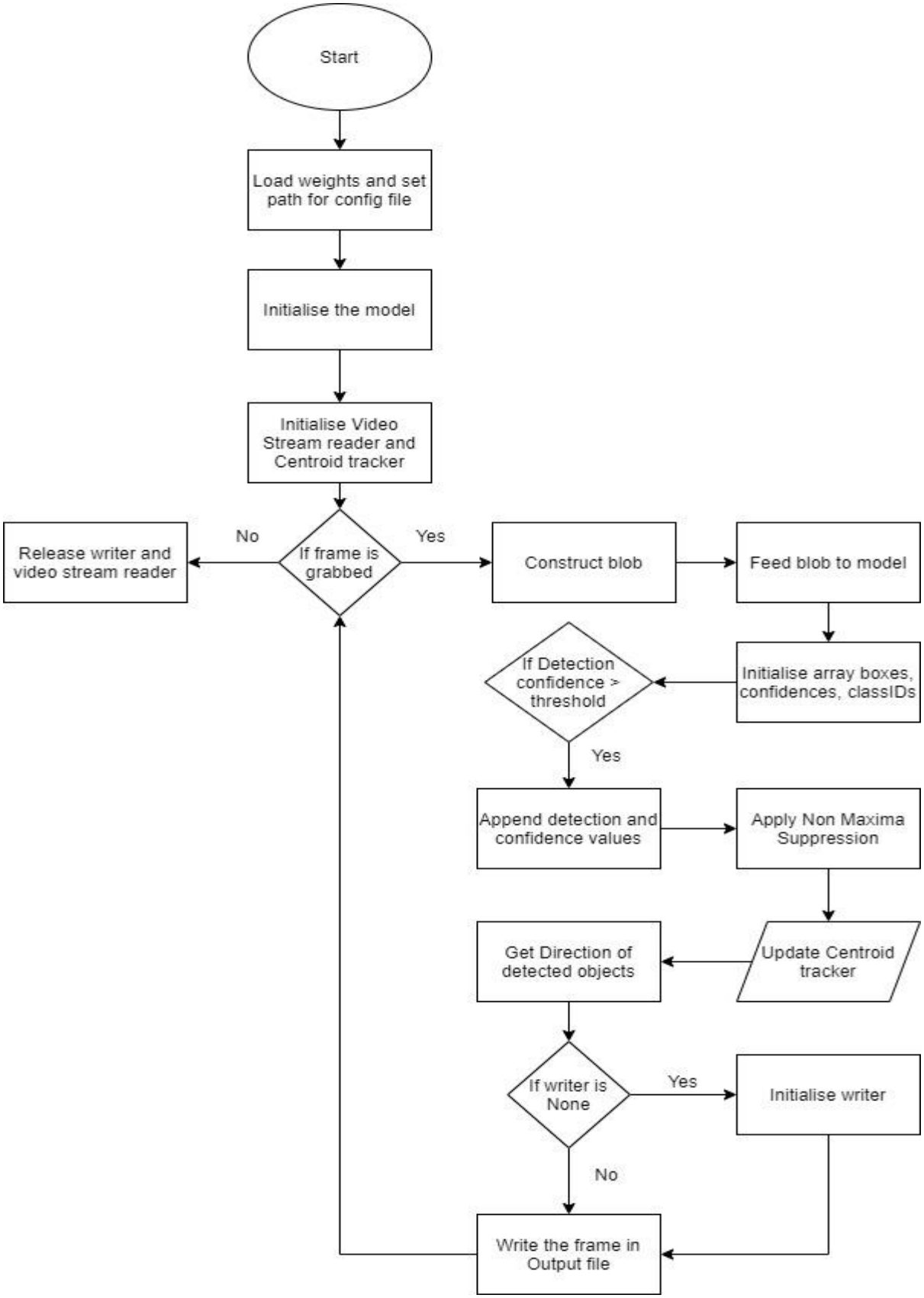

Fig7. Flowchart for Object Tracking Algorithm

Technical Needs For Real-Time Implementation

Deployment of the model is easy. All the necessary requirements are:

1. Cameras are to be installed at different parts inside the dock. The orientation needs to be such that the vehicle is clearly visible from the angle of camera along with the obstructions (people, things, etc) at least 240p for proper detection. The model can be tweaked according to the placement of camera to acquire proper distance metric for alert.
2. Our model should be deployed in a server at backend which should have wired or wireless connection to the camera for capturing video stream and real time detection. But as such a dedicated server is not required because of lightweight yolo v4 tiny.
3. The input directory for video stream can be provided manually to detect from the camera of choice or multiple instances of the model can be initialised to get input from different camera.
4. In the alert function of the model, only alert messages are enabled. A list of contact numbers for respective authorities is required to send the alert messages. For alerting the driver, either a common alarm can be setup on the dock or alert notification can be sent to driver or both. Again, the alert given to driver can be through mobile alert notification or some sort of alarm fixed inside truck.
5. Similarly, he object-tracker can be initialised and asked to make predictions from input taken by camera installed near the entrance. The model can be again modified to detect direction as in and out instead of left/right by changing few lines of code.
6. Both the algorithms heavily depend on the angle from which the identify the truck. If the model is given full back-view of the truck then they may not be able to identify the vehicle because of its resemblance to other materials involved in classification objective of algorithm. Either the model can be customised or angle of camera can be fitted in such a way that at least partial side of the object with hint of tyres are visible.

CHAPTER 5
CODING

Object Detection:

```
from google.colab import drive
drive.mount('/content/gdrive')

import numpy as np
import imutils
import argparse
import time
import cv2
import os
import time
import requests
import json
from time import sleep
from google.colab.patches import cv2_imshow
from scipy.spatial import distance

#HELPER FUNCTIONS
def TruckDetected(idx, L):
  for i in idx.flatten():
    if "truck" in L[classIDs[i]]:
      return True
    else:
      return False

def PeopleDetected(idx, L):
  for i in idx.flatten():
    if "person" in L[classIDs[i]]:
      return True
    else:
      return False

def InLine(nsd, classIDs, L, boxes):
  if nsd:
```

```
    for i in nsd:
      if "truck" in L[classIDs[i]]:
        flag = True
        idx = i
        break
      else:
        flag = False
    obj_inline_tr = []
    if flag:
      for i in nsd:
        if (mid_list[i][0] >= boxes[idx][0]) and (mid_list[i][0] <= boxes[idx][0] + boxes[idx][2]) :
          obj_inline_tr.append(i)
    return obj_inline_tr
  else:
    return []


def compute_distance(mid_list, ppm):
  dist = []
  nsd = []
  for i in range(0, len(mid_list)-1):
      for k in range(1, len(mid_list)):
          if k == i:
              break
          else:
              dst = distance.euclidean(mid_list[k], mid_list[i])/ppm
              dist.append(dst)
              if dst < 50 :
                  nsd.append(i)
                  nsd.append(k)
              nsd = list(dict.fromkeys(nsd))
  return dist, nsd

def sendSMS():
    url = "https://www.fast2sms.com/dev/bulkV2"
    '''
    my_data = {
        #Default Sender ID
```

```
        'sender_id': 'TXTIND',
        'message': 'Alert!!! Obstruction Nearby',
        'language': 'english',
        'route': 'v3',
        'numbers': '9452070851, 9161016378',
        'flash' : '0'
    }
    '''
    headers = {
        'authorization': 'L9Yu9syrN9oiIozIeW5OzRBiWR98Hc5O1D9lLSxCrVxFmpb5Ntu9nXOYbVFZ',
        'Content-Type': "application/x-www-form-urlencoded",
        'Cache-Control': "no-cache",
    }
    payload = "sender_id=TXTIND&message=Alert!!! Obstruction Nearby&route=v3&numbers=94520708
51,9161016378"

    response = requests.request("POST",
                            url,
                            data = payload,
                            headers = headers)
    time.sleep(3)
    print(response.text)



# Loading labels on which model was trained on
labelsPath = "/content/gdrive/MyDrive/Yolo/cfg/coco.names"
LABELS = open(labelsPath).read().strip().split("\n")
# initialize a list of colors to represent each possible class label
np.random.seed(42)
COLORS = np.random.randint(0, 255, size=(len(LABELS), 3),
  dtype="uint8")


#Path to weights and configuration files
#weightsPath = "/content/gdrive/MyDrive/Yolo/yolov4.weights"
weightsPath = "/content/gdrive/MyDrive/Yolo/yolov4-tiny.weights"
#configPath = "/content/darknet/cfg/yolov4.cfg"
configPath = "/content/darknet/cfg/yolov4-tiny.cfg"
```

```
print("[INFO] Loading YOLO")
CVnet = cv2.dnn.readNetFromDarknet(configPath, weightsPath)
lnD = CVnet.getLayerNames()
lnD = [lnD[i[0] - 1] for i in CVnet.getUnconnectedOutLayers()]



#Initialising Video Stream Reader
vsreader = cv2.VideoCapture("/content/yt1s.com - Indian Heavy Duty Trucks Are Running Live On Road
Truck Videos Entertainment World_v144P.mp4")
vswriter = None
(W, H) = (None, None)
#Trying to determine the total number of frames in the video file
try:
  prop = cv2.cv.CV_CAP_PROP_FRAME_COUNT if imutils.is_cv2() \
    else cv2.CAP_PROP_FRAME_COUNT
  total = int(vsreader.get(prop))
  print("[INFO] {} total frames in video".format(total))
#If error in counting total number of frames n video file
except:
  print("[INFO] could not determine # of frames in video")
  print("[INFO] no approx. completion time can be provided")
  total = -1

#Loop over frames from the video file stream
while True:
  (grabbed, frame) = vsreader.read()

  if not grabbed:
    break

  if W is None or H is None:
    (H, W) = frame.shape[:2]

  blob = cv2.dnn.blobFromImage(frame, 1 / 255.0, (416, 416),
    swapRB=True, crop=False)
  CVnet.setInput(blob)
  start = time.time()
  layerOutputs = CVnet.forward(ln)
```

```
end = time.time()

boxes = []
confidences = []
classIDs = []
mid_list = []
for output in layerOutputs:
  for detection in output:
    scores = detection[5:]
    classID = np.argmax(scores)
    confidence = scores[classID]
    #Filtering out weak predictions
    if confidence > 0.5:
      box = detection[0:4] * np.array([W, H, W, H])
      (centerX, centerY, width, height) = box.astype("int")
      x = int(centerX - (width / 2))
      y = int(centerY - (height / 2))
      mid_list.append((centerX, centerY))
      boxes.append([x, y, int(width), int(height)])
      confidences.append(float(confidence))
      classIDs.append(classID)

idxs = cv2.dnn.NMSBoxes(boxes, confidences, 0.5, 0.3)

#Defining Pixel-Per-Metric ratio
ppm = 0.5
dist, nsd = compute_distance(mid_list, ppm)
tr = InLine(nsd, classIDs, LABELS, boxes)

if len(idxs)>0:
  flagTr = TruckDetected(idxs, LABELS)
  flagHu = PeopleDetected(idxs, LABELS)
  if flagTr or flagHu:
    for i in idxs.flatten():
      (x, y) = (boxes[i][0], boxes[i][1])
      (w, h) = (boxes[i][2], boxes[i][3])

      if flagTr and (not flagHu):
```

```
        if (i in tr) and ("person" not in LABELS[classIDs[i]]):
          color = (0, 0, 255)
          cv2.rectangle(frame, (x, y), (x+w, y+h), color, 2)
          text = "Alert"
          cv2.putText(frame, text, (x,y-5), cv2.FONT_HERSHEY_SIMPLEX, 1, color, 2)
          _ = cv2.circle(frame, mid_list[i], 5, (255, 0, 0), -1)
          sendSMS()
          break
        #Case for which person is helping the truck
        #elif (i in tr) and ("person" in LABELS[classIDs[i]]):
        if (i not in nsd):
          color = [int(c) for c in COLORS[classIDs[i]]]
          cv2.rectangle(frame, (x, y), (x + w, y + h), color, 2)
          text = "{}: {:.4f}".format(LABELS[classIDs[i]],
            confidences[i])
          cv2.putText(frame, text, (x, y - 5),
            cv2.FONT_HERSHEY_SIMPLEX, 0.5, color, 2)
    else:
      for i in idxs.flatten():
        (x, y) = (boxes[i][0], boxes[i][1])
        (w, h) = (boxes[i][2], boxes[i][3])
        color = [int(c) for c in COLORS[classIDs[i]]]
        cv2.rectangle(frame, (x, y), (x + w, y + h), color, 2)
        text = "{}: {:.4f}".format(LABELS[classIDs[i]],
          confidences[i])
        cv2.putText(frame, text, (x, y - 5),
          cv2.FONT_HERSHEY_SIMPLEX, 0.5, color, 2)

    if vswriter is None:
      #Initialize video writer
      fourcc = cv2.VideoWriter_fourcc(*"MJPG")
      vswriter = cv2.VideoWriter("/content/out2.mp4", fourcc, 30,
        (frame.shape[1], frame.shape[0]), True)                #args["output"]
      # some information on processing single frame
    if total > 0:
      elap = (end - start)
      print("[INFO] single frame took {:.4f} seconds".format(elap))
      print("[INFO] estimated total time to finish: {:.4f}".format(
```

```
        elap * total))
    #cv2_imshow(frame)
    vswriter.write(frame)

#Release the file pointers
print("[INFO] cleaning up...")
vswriter.release()
vsreader.release()
```

Object Tracking:

```
from google.colab import drive
drive.mount('/content/gdrive')

!pip install opencv-contrib-python==3.4.13.47 --force-reinstall

from scipy.spatial import distance as dist
from google.colab.patches import cv2_imshow
from __future__ import division, print_function, absolute_import
from collections import OrderedDict
from imutils.video import VideoStream
from collections import deque, OrderedDict
import generate_detections as gdet
import numpy as np
import argparse
import imutils
import time
import cv2
import preprocessing
import nn_matching
import matplotlib.pyplot as plt

class CentroidTracker():
  def __init__(self, maxDisappeared=5, buffer = 3):
    """
    Info: Initialise unique Object ID and Queue buffer and track centroid, directon of each object and consec
utive frames object has disappeared
    Params:
```

```
    1. maxDisappeared - Number of consecutives frames the object is missing
    2. buffer - Buffer for queue
    """
    self.nextObjectID = 0
    self.buffer = buffer
    self.objects = OrderedDict()
    self.disappeared = OrderedDict()
    self.pts = OrderedDict()
    self.Dir = OrderedDict()
    self.maxDisappeared = maxDisappeared

  def register(self, centroid):
    """
    Info: Assign Unique next ID to newly detected object
    Params:
    1. centroid - Centroid of detected object
    """
    self.objects[self.nextObjectID] = centroid
    self.pts[self.nextObjectID] = deque(maxlen = self.buffer)
    self.pts[self.nextObjectID].appendleft(centroid)
    self.Dir[self.nextObjectID] = None
    self.disappeared[self.nextObjectID] = 0
    self.nextObjectID += 1
  def deregister(self, objectID):
    """
    Info: Delete Objects which are not in consecutive frames for specified number of times
    Params:
    1. ObjectID: Object ID of detected to deregister
    """
    del self.objects[objectID]
    del self.disappeared[objectID]
    del self.pts[objectID]
    del self.Dir[objectID]

  def update(self, rects):
    """
    Info: Register, Deresgister or update object Ids and centroids
    Params:
```

```
1. rects: Bounding box of detected objects
"""
if len(rects) == 0:
  for objectID in list(self.disappeared.keys()):
    self.disappeared[objectID] += 1
    if self.disappeared[objectID] > self.maxDisappeared:
      self.deregister(objectID)
  return self.objects, self.Dir

inputCentroids = np.zeros((len(rects), 2), dtype="int")
for (i, (startX, startY, w, h)) in enumerate(rects):
  cX = int(startX + (w / 2.0))
  cY = int(startY + (h / 2.0))
  inputCentroids[i] = (cX, cY)

if len(self.objects) == 0:
  for i in range(0, len(inputCentroids)):
    self.register(inputCentroids[i])
else:
  objectIDs = list(self.objects.keys())
  objectCentroids = list(self.objects.values())
  D = dist.cdist(np.array(objectCentroids), inputCentroids)
  rows = D.min(axis=1).argsort()
  cols = D.argmin(axis=1)[rows]
  usedRows = set()
  usedCols = set()
  for (row, col) in zip(rows, cols):
    threshold = 0
    if row in usedRows or col in usedCols:
      continue
    objectID = objectIDs[row]
    self.objects[objectID] = inputCentroids[col]
    self.pts[objectID].appendleft(inputCentroids[col])
    self.disappeared[objectID] = 0
    usedRows.add(row)
    usedCols.add(col)
  unusedRows = set(range(0, D.shape[0])).difference(usedRows)
  unusedCols = set(range(0, D.shape[1])).difference(usedCols)
```

```
    if D.shape[0] >= D.shape[1]:
      for row in unusedRows:
        objectID = objectIDs[row]
        self.pts[objectID].appendleft(None)
        self.disappeared[objectID] += 1
        if self.disappeared[objectID] > self.maxDisappeared:
          self.deregister(objectID)
      else:
        for col in unusedCols:
          self.register(inputCentroids[col])

      #Finding Directions
      for j in (self.pts.keys()):
        pts = self.pts[j]
        for i in np.arange(1, len(pts)):
          if pts[i - 1] is None or pts[i] is None:
            continue
          if len(pts) >= self.buffer and i == 1 and pts[-1*self.buffer] is not None:
            dX = pts[-1*self.buffer][0] - pts[i][0]
            dY = pts[-1*self.buffer][1] - pts[i][1]
            (dirX, dirY) = ("", "")
            if np.abs(dX) > 5:
              dirX = "Right" if np.sign(dX) == 1 else "Left"
            if np.abs(dY) > 20:
              dirY = "Up" if np.sign(dY) == 1 else "Down"
            if dirX != "" and dirY != "":
              self.Dir[j] = "{}-{}".format(dirY, dirX)
            else:
              self.Dir[j] = dirX if dirX != "" else dirY
    return self.objects, self.Dir

# Loading labels on which model was trained on
labelsPath = "/content/gdrive/MyDrive/Yolo/cfg/coco.names"
LABELS = open(labelsPath).read().strip().split("\n")
#print(LABELS)
# initialize a list of colors to represent each possible class label
np.random.seed(42)
```

```
COLORS = np.random.randint(0, 255, size=(len(LABELS), 3),
  dtype="uint8")

#Paths to weights and configuration files
#weightsPath = "/content/gdrive/MyDrive/Yolo/yolov4.weights"
weightsPath = "/content/gdrive/MyDrive/Yolo/yolov4-tiny.weights"
#configPath = "/content/darknet/cfg/yolov4.cfg"
configPath = "/content/darknet/cfg/yolov4-tiny.cfg"
print("[INFO] loading YOLO")
CVnet = cv2.dnn.readNetFromDarknet(configPath, weightsPath)
lnD = CVnet.getLayerNames()
lnD = [lnD[i[0] - 1] for i in CVnet.getUnconnectedOutLayers()]

#Initialising video stream reader
vsreader = cv2.VideoCapture("/content/Video2.mp4")
vswriter = None
(W, H) = (None, None)
# Try to determine the total number of frames in the video file
try:
  prop = cv2.cv.CV_CAP_PROP_FRAME_COUNT if imutils.is_cv2() \
    else cv2.CAP_PROP_FRAME_COUNT
  total = int(vs.get(prop))
  print("[INFO] {} total frames in video".format(total))
#If error in counting number of frames
except:
  print("[INFO] could not determine # of frames in video")
  print("[INFO] no approx. completion time can be provided")
  total = -1

ct = CentroidTracker()
(H, W) = (None, None)
#Loop over frames from the video file stream
while True:
  (grabbed, frame) = vs.read()
  if not grabbed:
    break
  if W is None or H is None:
    (H, W) = frame.shape[:2]
```

```
blob = cv2.dnn.blobFromImage(frame, 1 / 255.0, (416, 416),
  swapRB=True, crop=False)
CVnet.setInput(blob)
start = time.time()
layerOutputs = CVnet.forward(ln)
end = time.time()

boxes = []
confidences = []
clsIDs = []
mid_list = []
for output in layerOutputs:
  for detection in output:
    scores = detection[5:]
    classID = np.argmax(scores)
    confidence = scores[classID]
    #Filtering out weak prediction
    if confidence > 0.5:
      box = detection[0:4] * np.array([W, H, W, H])
      (centerX, centerY, width, height) = box.astype("int")
      x = int(centerX - (width / 2))
      y = int(centerY - (height / 2))
      mid_list.append((centerX, centerY))
      boxes.append([x, y, int(width), int(height)])
      confidences.append(float(confidence))
      clsIDs.append(classID)

idxs = cv2.dnn.NMSBoxes(boxes, confidences, 0.5, 0.3)
if len(idxs)>0:
  for i in idxs.flatten():
      (x, y) = (boxes[i][0], boxes[i][1])
      (w, h) = (boxes[i][2], boxes[i][3])
      color = [int(c) for c in COLORS[clsIDs[i]]]
      cv2.rectangle(frame, (x, y), (x + w, y + h), color, 2)

      text = "{}: {:.4f}".format(LABELS[clsIDs[i]],
        confidences[i])
      cv2.putText(frame, text, (x, y - 5),
```

```
        cv2.FONT_HERSHEY_SIMPLEX, 0.5, color, 2)
    objects, Dir = ct.update(boxes)
    for (objectID, centroid) in objects.items():
      if Dir[objectID] is not None:
        text = "ID {} - {}".format(objectID, Dir[objectID])
        cv2.putText(frame, text, (centroid[0] - 10, centroid[1] - 10),
          cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 1)
        cv2.circle(frame, (centroid[0], centroid[1]), 4, (0, 255, 0), -1)
      else:
        text = "ID {}".format(objectID)
        cv2.putText(frame, text, (centroid[0] - 10, centroid[1] - 10),
          cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 1)
        cv2.circle(frame, (centroid[0], centroid[1]), 4, (0, 255, 0), -1)
    if vswriter is None:
      #Initialize video writer
      fourcc = cv2.VideoWriter_fourcc(*"MJPG")
      vswriter = cv2.VideoWriter("/content/output.mp4", fourcc, 30,
        (frame.shape[1], frame.shape[0]), True)
    if total > 0:
      elap = (end - start)
      print("[INFO] single frame took {:.4f} seconds".format(elap))
      print("[INFO] estimated total time to finish: {:.4f}".format(
        elap * total))
    #cv2_imshow(frame)
    vswriter.write(frame)
#Release the file pointers
print("[INFO] cleaning up...")
vswriter.release()
vsreader.release()
```

CHAPTER 6
RESULTS AND ANALYSIS

Some snapshots of the result are:

Fig. 8 Alert Message Output

Fig. 9 Closeness Of Truck detected by Algorithm

Fig. 10 Direction Detection By Object Tracking Algorithm

The improved version of the object detection code contains a couple of use cases. They are:

1.) First one is, we try to detect the objects that are in line i.e. in the path of the truck. Since the alarm should not go off when anything is close to truck but not in its paths and hence no mishap.
2.) Second one is, we also try to remove stationary truck from detection as it would not contribute to accidents

Drive Link for output videos:
https://drive.google.com/file/d/1CAT2SV3xZDLEbiL303FVYeI__l1gzkZ2/view?usp=sharing
https://drive.google.com/file/d/1X-6vZh1vlfIy83ledgeL1NQCKccnwo-8/view?usp=sharing

COMAPRING WITH EXISTING PRODUCTS AND PROTOTYPES

Although many third-party services have started providing object detection prototypes through cloud infrastructures, they also charge for it. The base code remains with the third party and hence charges incur. The main of this prototype is:

- Use of open-source implementation for object detection that is reliable, flexible and easily deployable.
- If the alarming system is implemented inside the dock through a LAN then there is no third party intervention other than for internet connection.
- The logic and safety measures can be defined to any place by just tweaking the input parameters.
- This product ensures safety and doesn't require in-person surveillance throughout it's working

PROS AND CONS

Pros:

- The model ensures automation of safety at dock by detecting obstructions and engaging alert notifications.
- No need for third party intervention.
- Algorithms is based on state-of-the-art techniques used for object detection and hence provides faster working. The algorithm provides frame processing speed of 440 fps under darknet infrastructure with GPU.

- Barring the camera installation costs, wired or wireless connection and server for deployment, no extra cost to be expensed.
- Flexibility of logic is provided which can make distance metric prediction dynamic or manual according to needs of user.

Cons:

- In direction detection of moving vehicle, if object couldn't be detected properly, multiple IDs can be assigned to same object. This may result from poor camera resolution or improper orientation due to which the object cannot be detected. Due to this direction can't be predicted as buffer needs to be filled before predicting it.
- Same condition applies for object detection. If the object is not detected as a truck for some consecutive number of frames, then there won't be any alert for those consecutive number of frames.
- Our model highly depends on the positioning of camera as shown in Fig. 11. A truck's path should be visible from the view and moreover it would be much better if side view the truck is facing the camera even if partially. The model fails to detect the truck from full back-view, hence it is recommended to customize the model with respect to camera and distance metric as proposed that is suitable for the application.

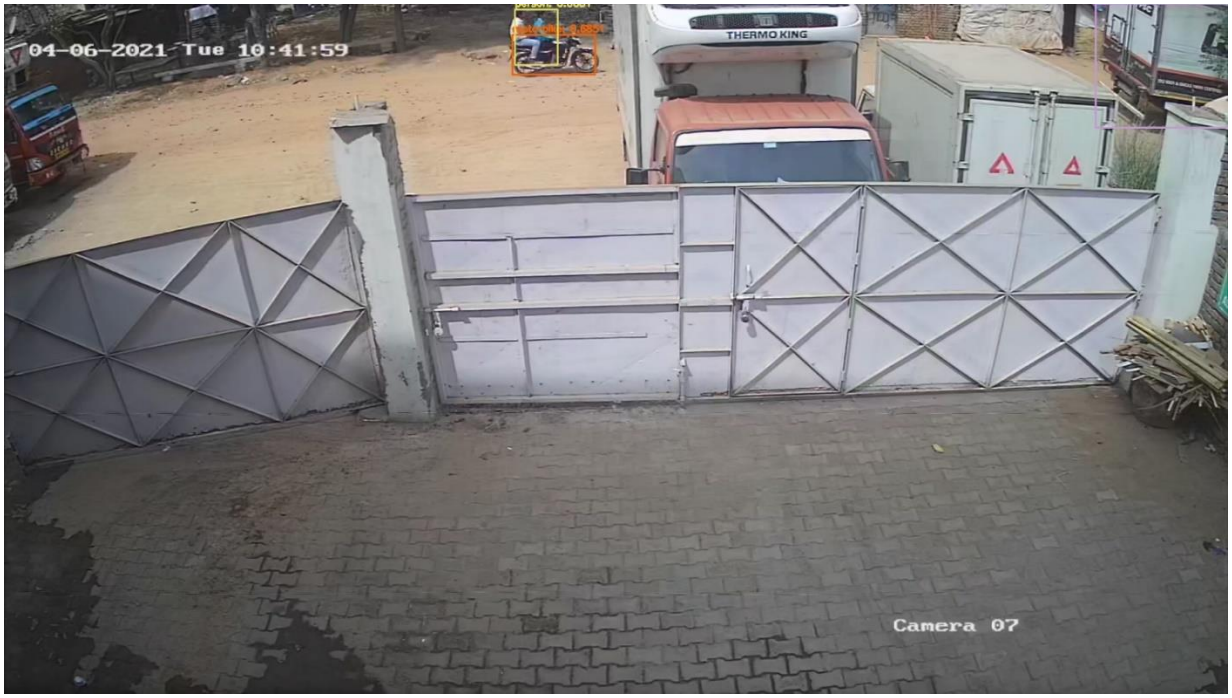

Fig. 11 Camera Angle Dependency

CHAPTER 7
CONCLUSION

The proposed model uses YOLOv4 for object detection and object tracking. The object tracking algorithm uses the concept of centroid tracker to assign unique IDs to all the vehicles and detect their direction of motion. The logic is also provided for removing orientation of camera from distance perspective i.e. to map SI unit of distance to pixel distance/length of object in the image. Several use cases have been tried to be covered for preventing unwanted scenarios for alert notification.

The future work may include extending our model logic to several other industrial use cases before deploying our model. Some of them are:

- Differentiating workers who are unaware of the vehicle being reversed and those who are helping in unloading or reversing the truck. This can be done by including gesture detection of the person in-line with the trucks path.
- Other area of improvisation could be tagging inanimate objects and setting maximum frame limit so that the alarm notification goes off only when the obstruction is detected for consecutive number of frames as the driver maybe aware of obstruction but could need some time and space to get away from obstruction.
- Image segmentation can also be tried to reduce the risk of bigger bounding box in unfamiliar cases to the algorithm.

CHAPTER 8
BIBLIOGRAPHY

1. B.-I. Kim, R. J. Graves, S. S. Heragu, and A. S. Onge, "Intelligent agentmodeling of an industrial warehousing problem," IIE Transactions, Article vol. 34, no. 7, p. 601, 2002.
2. F. Hamdi, A. Ghorbel, F. Masmoudi, and L. Dupont, "Optimization of a supply portfolio in the context of supply chain risk management: literature review," Journal of Intelligent Manufacturing, vol. 29, no. 4, pp. 763-788, 2018.
3. J.Dou, C. Chen, and P. Yang, "Genetic scheduling and reinforcement learning in multirobot systems for intelligent warehouses," Mathematical Problems in Engineering, Article pp. 1-10, 2015.
4. R. Hlioui, A. Gharbi, and A. Hajji, "Replenishment, production and quality control strategies in the three-stage supply chain," International Journal of Production Economics, vol. 166, p. 90, 2015.
5. S. Bandaru, T. Aslam, A. Ng, and K. Deb, "Generalized higher-level automated innovation with application to inventory management," European Journal of Operational Research, vol. 243, no. 2, p. 480, 2015.
6. S. Tereza, "A suitable artificial intelligence model for inventory level optimization," Trendy Ekonomiky a Managementu, vol. 10, no. 25, pp. 48-55, 2016.
7. X.-Y. Gong, H. Su, D. Xu, Z.-T. Zhang, F. Shen, and H.-B. Yang, "An overview of contour detection approaches," International Journal of Automation and Computing, vol. 15, no. 6, p. 656, 2018.