



Department of Computer Science and Engineering College of
Engineering, Guindy
Anna University, Chennai – 600 025

ESTIMATION OF THE SPEED OF DRONES FROM AERIAL FOOTAGE

MACHINE LEARNING PROJECT

NAME	REG.NO
Shreya Ananth	2019103580
Sneha S	2019103583

1. ABSTRACT

In today's world, video surveillance has become a new normal. Geographical expanses of land can be monitored and surveyed by cameras that are attached to drones or other manned vehicles like helicopter and airplanes. They can be used for a lot of purposes like forest fire monitoring, aerial photography, product deliveries, agriculture, policing and surveillance, and infrastructure inspections. They can also be used for military purposes. Estimating the speed at which the camera moved at the time of video capturing can have a lot of advantages. From an aerial footage video, we can even route back to the vehicle that was used for capturing videos, by narrowing down on the air traffic logs. The speed estimated from the videos that are live transmitted from manned air vehicles, can be used to detect pilot's unresponsiveness in case of medical emergencies or hijacks. The estimation of speed is done by the detection of buildings and trees in the visible areas, these can also be used to obtain abundant information such as the number of buildings, tree population and so on from video image sequences that have low cost and high efficiency. Detection from video is a hot research field for computer vision. There are a lot of advancements to detect moving targets from a static source. In this paper, the estimation of the speed of a moving source by detecting the targets in the footage is presented.

2. INTRODUCTION

The aim of this project is to estimate the speed of cameras from aerial footage. These can have a lot of advantages and is a state-of-the-art problem. In this project, we have approached the above-mentioned problem by first detecting the objects that will be static in a real-world point of view. Images taken by UAVs or drones are quite different from images taken by using a normal camera. For that reason, it cannot be assumed that object detection algorithms normally used on "normal" images perform well on taken by drone images. This is a tedious task as the video

from a moving camera will have both the static objects and the dynamic objects will have everything in motion. Thus, to isolate the static components from their dynamic counterparts, we detect objects like trees and buildings. This is done with the help of using Tensorflow object detection API. After detecting the static objects, we need to tokenize them to track their movements. The speed at which the camera moves will be the same as the speed of the static objects in the opposite direction. Detecting the motion of the object between two frames will provide us the movement in pixel magnitude. Calculating the ppm (pixels per metre) will help us in obtaining the speed provided we know the number of frames per second.

2.1 PROBLEM STATEMENT:

The main objective of this project is to estimate the speed at which a camera moves from the video footage that was taken from air. This faces a problem as the images captured by a drone often are different from those available for training, which are often taken by a hand-held camera. Difficulties in detecting objects in data from a drone may arise due to the positioning of the camera compared to images taken by a human, depending on what type of images the network is trained on.

2.2 PROBLEM SOLUTION:

DOMAIN: Computer Vision

The computer vision technique to detect object such as buildings and trees plays a major role in solving this particular project. The tracking of several static (moving in video) are done with the help of correlation tracker from the dlib library. The speed is estimated by calculating the amount of pixels traversed and the ppm(pixels per meter)

2.3 DATASET

The aerial footage videos are acquired from YouTube to find the speed at which the drone moves.

3. LITERATURE SURVEY

The estimation of camera speed with a moving source is done by combining object detection, object tracking and speed computation.

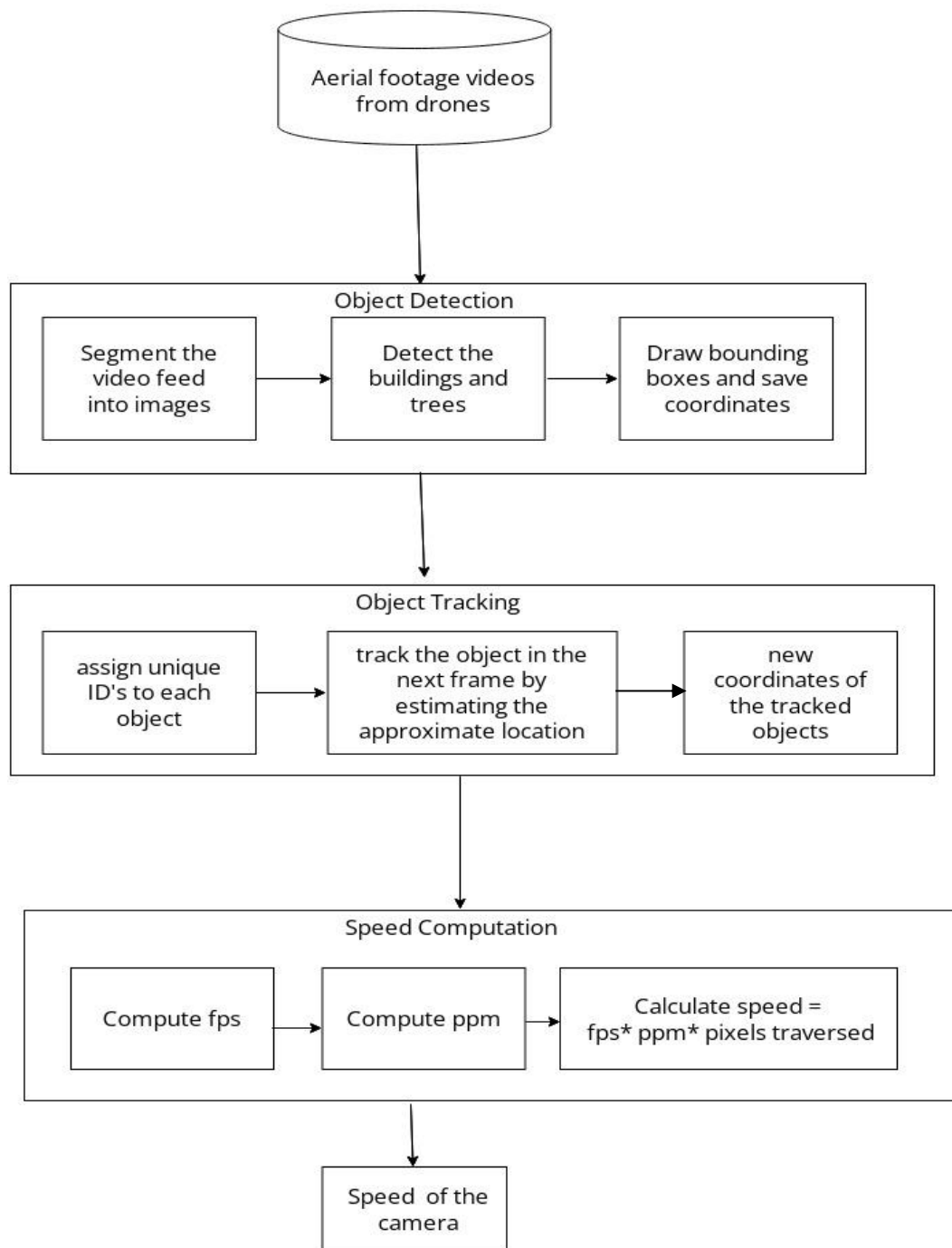
Sun, C., Zhan, W., She, J. and Zhang, Y,[1] came up with a method to detect the presence of buildings, trees, persons and cars from a video footage. This method was trained with the help of convolutional neural networks. Migration learning and Tensorflow API with pretrained models were used to accomplish the task of detecting buildings and trees.

Singh, Upasna, Anjali Saini, and Ravi Domala [2] used CNN to track the movement of objects. This used two Inception V3 architectures that contain the CNN framework. Inception V3 has shown to outperform other architectures for object classification tasks. This method uses extensive offline training of neural networks for tracking different objects at high speed in real time.

J. Wang [3] presented a vehicle speed estimation algorithm based on moving target detection in video surveillance. Firstly, the features of moving vehicles were extracted by using the three-frame difference method and the background difference method; secondly, tracked and positioned the moving target according to moving vehicle centroid feature extraction method; finally, the vehicle speed was estimated based on mapping relationship between the pixel distance with the actual distance.

4. SYSTEM DESIGN

a. ARCHITECTURAL DIAGRAM



5. MODULE EXPLANATION:

a. Object Detection

In this module, for each frame of the video that is segmented as an image, the buildings and trees that are present in the frame are detected and are covered with bounding boxes. The detected buildings and trees are categorized into classes with labels 1, 2 assigned to them. 1 denotes the trees and 2 denotes the classes.

Input – A frame that is segmented from the video

Output – The frame that contains as many bounding boxes as the number of buildings and trees present, the coordinates of each bounding box.

b. Object Tracking

This module accomplishes two tasks,

- assign unique IDs to each object in an image
- track each of the objects and associated IDs as they move around in a video stream.

The dlib correlation tracker is used to track the movement of buildings and trees in the video. It uses both the prior information regarding the location of the object bounding box in the previous frame along with data garnered from the current frame to infer where the new location of the object is.

Input – The frame and the coordinates of a detected object

Output – The newly tracked coordinates of the tracked objects.

c. Speed Computation

The Speed computation is done instantaneously within 2 successive frames by multiplying the pixel distance between the object in two different frames and the ppm (pixels per meter). The ppm is estimated by acquired the width of the road from google maps and dividing by the number of pixels that span the width of the road. For the entire stretch of the video, the speed between two frames is now multiplied by fps (frames per second). This estimates the

speed in m/s. When multiplied with 3.6, the speed of the camera is estimated in km/hr.

Input – ppm, distance travelled in pixels between 2 frames, fps

Output – Speed traveled by the camera.

6. IMPLEMENTATION

- Initial Setup:

Downloading dataset and setting up Jupyter notebook.

- Dataset:

Various aerial surveillance videos taken from a drone from YouTube.

- Tools:

Jupyter notebook

Installing necessary dependencies:

```
[1] %cd /content/drive/MyDrive/Test

/content/drive/MyDrive/Test

[2] !git clone https://github.com/cocodataset/cocoapi.git

fatal: destination path 'cocoapi' already exists and is not an empty directory.

[3] %cd cocoapi/PythonAPI

/content/drive/MyDrive/Test/cocoapi/PythonAPI

!make

skipping 'pycocotools/_mask.c' Cython extension (up-to-date)
building 'pycocotools._mask' extension
creating build
creating build/common
creating build/temp.linux-x86_64-3.7/pycocotools
x86_64-linux-gnu-gcc -pthread -Wno-unused-result -Wsign-compare -DNDEBUG -g -fwrapv -O2 -Wall -g -fstack-protector-strong -Wformat -Werror=format-security -g -fwrapv -O2 -g
../common/maskApi.c: In function 'rleDecode':
../common/maskApi.c:166:3: warning: this 'for' clause does not guard... [-Wmisleading-indentation]
    for( j=0; j<k; j++) x[j]=(int)(scale*xy[j*2+0]+.5); x[k]=x[0];
    ^~~
../common/maskApi.c:167:54: note: ...this statement, but the latter is misleadingly indented as if it were guarded by the 'for'
    for( j=0; j<k; j++) x[j]=(int)(scale*xy[j*2+0]+.5); x[k]=x[0];
    ^~~
../common/maskApi.c:166:3: warning: this 'for' clause does not guard... [-Wmisleading-indentation]
    for( j=0; j<k; j++) y[j]=(int)(scale*xy[j*2+1]+.5); y[k]=y[0];
    ^~~
../common/maskApi.c:167:54: note: ...this statement, but the latter is misleadingly indented as if it were guarded by the 'for'
    for( j=0; j<k; j++) y[j]=(int)(scale*xy[j*2+1]+.5); y[k]=y[0];
    ^~~

Executing (4s) Cell > system() > _system_compat() > _run_command() > _monitor_process() > _poll_process()
```



```

[5] %cd /usr/local/lib/python3.7/dist-packages/tensorflow
/usr/local/lib/python3.7/dist-packages/tensorflow

[6] !git clone https://github.com/tensorflow/models.git
fatal: destination path 'models' already exists and is not an empty directory.

[7] %cd /content/drive/MyDrive/Test/cocoapi/PythonAPI
/content/drive/MyDrive/Test/cocoapi/PythonAPI

[8] cp -r pycocotools /usr/local/lib/python3.7/dist-packages/tensorflow/models/research/

[9] %cd /usr/local/lib/python3.7/dist-packages/tensorflow/models/research
/usr/local/lib/python3.7/dist-packages/tensorflow/models/research

[10] !protoc object_detection/protos/*.proto --python_out=.

[11] !export PYTHONPATH=$PYTHONPATH:`pwd`:`pwd`/slim

[12] cp /content/drive/MyDrive/Test/video.mp4 /usr/local/lib/python3.7/dist-packages/tensorflow/models/research/object_detection

[13] cp -r /content/drive/MyDrive/Test/g3doc /usr/local/lib/python3.7/dist-packages/tensorflow/models/research/object_detection

```

Executing (0s) Cell > <module> > <module> > <module>

```

[7] %cd /content/drive/MyDrive/Test/cocoapi/PythonAPI
/content/drive/MyDrive/Test/cocoapi/PythonAPI

[8] cp -r pycocotools /usr/local/lib/python3.7/dist-packages/tensorflow/models/research/

[9] %cd /usr/local/lib/python3.7/dist-packages/tensorflow/models/research
/usr/local/lib/python3.7/dist-packages/tensorflow/models/research

[10] !protoc object_detection/protos/*.proto --python_out=.

[11] !export PYTHONPATH=$PYTHONPATH:`pwd`:`pwd`/slim

[12] cp /content/drive/MyDrive/Test/video.mp4 /usr/local/lib/python3.7/dist-packages/tensorflow/models/research/object_detection

[13] cp -r /content/drive/MyDrive/Test/g3doc /usr/local/lib/python3.7/dist-packages/tensorflow/models/research/object_detection

Double-click (or enter) to edit

[14] cp -r /content/drive/MyDrive/Test/ckpt_faster_rcnn_inception /usr/local/lib/python3.7/dist-packages/tensorflow/models/research/object_detection

[15] cp -r /content/drive/MyDrive/Test/label_map /usr/local/lib/python3.7/dist-packages/tensorflow/models/research/object_detection

[16] pip install tf_slim
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: tf_slim in /usr/local/lib/python3.7/dist-packages (1.1.0)
Requirement already satisfied: absl-py>=0.2.2 in /usr/local/lib/python3.7/dist-packages (from tf_slim) (1.1.0)

[17] import numpy as np
import os

```

Executing (47s) Cell > trackMultipleObjects() > run() > _run() > _do_run() > _do_call() > _run_fn() > _call_tf_sessionrun()

a. Object Detection

Setting up the environment:

Importing major modules:

```
import numpy as np
import os
import sys
import tensorflow as tf
import cv2
from google.colab.patches import cv2_imshow
import cv2
import dlib
import time
import threading
import math
```

Changing file format to open in Google Colab:

```
from IPython.display import HTML
from base64 import b64encode
import os

# Input video path
save_path = '/usr/local/lib/python3.7/dist-packages/tensorflow/models/research/object_detection/video.mp4'

# Compressed video path
compressed_path = '/usr/local/lib/python3.7/dist-packages/tensorflow/models/research/object_detection/test2.mp4'

os.system(f'ffmpeg -i {save_path} -vcodec libx264 {compressed_path}')
```

256

OBJECT DETECTION:

Setting the values of global variables for object detection

```
[19] WIDTH = 1280
    HEIGHT = 720

[20] # This is needed since the notebook is stored in the object_detection folder.
    sys.path.append("..")
    from tensorflow.models.research.object_detection.utils import ops as utils_ops

    if tf.__version__ < '1.4.0':
        raise ImportError('Please upgrade your tensorflow installation to v1.4.* or later!')

[21] ### Object detection imports
    # Here are the imports from the object detection module.
    from tensorflow.models.research.object_detection.utils import label_map_util
    from tensorflow.models.research.object_detection.utils import visualization_utils as vis_util

[22] ## Model preparation
    MODEL_NAME = '/usr/local/lib/python3.7/dist-packages/tensorflow/models/research/object_detection/ckpt_faster_rcnn_inception'

[23] # Path to frozen detection graph. This is the actual model that is used for the object detection.
    PATH_TO_CKPT = MODEL_NAME + '/frozen_inference_graph.pb'

[24] # List of the strings that is used to add correct label for each box.
    PATH_TO_LABELS = os.path.join('/usr/local/lib/python3.7/dist-packages/tensorflow/models/research/object_detection/label_map', 'object_detection.pbtxt')
    NUM_CLASSES = 4
```

Loading the pre-existing Tensorflow model into memory:

```
[25] ### Load a (frozen) Tensorflow model into memory.
detection_graph = tf.compat.v1.Graph()
with detection_graph.as_default():
    od_graph_def = tf.compat.v1.GraphDef()
    with tf.compat.v2.io.gfile.GFile(PATH_TO_CKPT, 'rb') as fid:
        serialized_graph = fid.read()
        od_graph_def.ParseFromString(serialized_graph)
        tf.import_graph_def(od_graph_def, name='')

[26] ### Loading label map
label_map = label_map_util.load_labelmap(PATH_TO_LABELS)
categories = label_map_util.convert_label_map_to_categories(label_map, max_num_classes=NUM_CLASSES, use_display_name=True)
category_index = label_map_util.create_category_index(categories)
```

Calculating frames per second (fps)

```
[28] frameNr = 0
c=0
video = cv2.VideoCapture(compressed_path)
frame_count = video.get(cv2.CAP_PROP_FRAME_COUNT)
fps = video.get(cv2.CAP_PROP_FPS)
print("fps",fps)
print("Frame Count",frame_count)
duration = int(frame_count/fps)
print("duration",duration)          #duration

fps 29.97002997002997
Frame Count 393.0
duration 13
```

b. Object Tracking

Initializing the variables for tracking & updating the tracked objects:

```
def trackMultipleObjects():
    rectangleColor = (0, 255, 0)
    frameCounter = 0
    currentCarID = 0

    carTracker = {}
    carNumbers = {}
    carLocation1 = {}
    carLocation2 = {}
    speed = [None] * 1000

    with detection_graph.as_default():
        with tf.compat.v1.Session(graph=detection_graph) as sess:
            while True:
                start_time = time.time()
                rc, image = video.read()
                if type(image) == type(None):
                    break

                image = cv2.resize(image, (WIDTH, HEIGHT))
                resultImage = image.copy()
                frameCounter = frameCounter + 1

                carIDToDelete = []

                for carID in carTracker.keys():
                    trackingQuality = carTracker[carID].update(image)

                    if trackingQuality < 7:
                        carIDToDelete.append(carID)

                for carID in carIDToDelete:
                    carTracker.pop(carID, None)
                    carLocation1.pop(carID, None)
                    carLocation2.pop(carID, None)
```

Using the object detector to get the coordinates:

```

if not (frameCounter % 10):
    ops = tf.compat.v1.get_default_graph().get_operations()
    all_tensor_names = {output.name for op in ops for output in op.outputs}
    tensor_dict = {}
    for key in ['num_detections', 'detection_boxes', 'detection_scores', 'detection_classes', 'detection_masks']:
        tensor_name = key + ':0'
        if tensor_name in all_tensor_names:
            tensor_dict[key] = tf.compat.v1.get_default_graph().get_tensor_by_name(
                tensor_name)

    image_tensor = tf.compat.v1.get_default_graph().get_tensor_by_name('image_tensor:0')

    # Run inference
    output_dict = sess.run(tensor_dict,
                           feed_dict={image_tensor: np.expand_dims(image, 0)})

    # all outputs are float32 numpy arrays, so convert types as appropriate
    output_dict['num_detections'] = int(output_dict['num_detections'][0])
    output_dict['detection_classes'] = output_dict[
        'detection_classes'][0].astype(np.uint8)
    output_dict['detection_boxes'] = output_dict['detection_boxes'][0]
    output_dict['detection_scores'] = output_dict['detection_scores'][0]

    cars = []
    i = image.shape
    w = i[0]
    h = i[1]

    for i in range(len(output_dict['detection_boxes'])):
        if output_dict['detection_classes'][i]==2:
            bb = [int(output_dict['detection_boxes'][i][1]*w), int(output_dict['detection_boxes'][i][0]*h),
                  int((output_dict['detection_boxes'][i][3]-output_dict['detection_boxes'][i][1])*w),
                  int((output_dict['detection_boxes'][i][2]-output_dict['detection_boxes'][i][0])*h)]
            cars.append(bb)

```

Tracking and finding positions of existing objects & adding new objects:

```

for (_x, _y, _w, _h) in cars:
    x = int(_x)
    y = int(_y)
    w = int(_w)
    h = int(_h)

    x_bar = x + 0.5 * w
    y_bar = y + 0.5 * h

    matchCarID = None

    for carID in carTracker.keys():
        trackedPosition = carTracker[carID].get_position()

        t_x = int(trackedPosition.left())
        t_y = int(trackedPosition.top())
        t_w = int(trackedPosition.width())
        t_h = int(trackedPosition.height())

        t_x_bar = t_x + 0.5 * t_w
        t_y_bar = t_y + 0.5 * t_h

        if ((t_x <= x_bar <= (t_x + t_w)) and (t_y <= y_bar <= (t_y + t_h)) and (x <= t_x_bar <= (x + w)) and (y <= t_y_bar <= (y + h))):
            matchCarID = carID

    if matchCarID is None:
        tracker = dlib.correlation_tracker()
        tracker.start_track(image, dlib.rectangle(x, y, x + w, y + h))

        carTracker[currentCarID] = tracker
        carLocation[currentCarID] = [x, y, w, h]

        currentCarID = currentCarID + 1

```

Getting the tracked position of the objects & estimating their speeds:

```

for carID in carTracker.keys():
    trackedPosition = carTracker[carID].get_position()

    t_x = int(trackedPosition.left())
    t_y = int(trackedPosition.top())
    t_w = int(trackedPosition.width())
    t_h = int(trackedPosition.height())

    # speed estimation
    carLocation2[carID] = [t_x, t_y, t_w, t_h]

for i in carLocation1.keys():
    if frameCounter % 1 == 0:

        [x1, y1, w1, h1] = carLocation1[i]
        [x2, y2, w2, h2] = carLocation2[i]

        carLocation1[i] = [x2, y2, w2, h2]
        if [x1, y1, w1, h1] != [x2, y2, w2, h2]:
            if (speed[i] == None or speed[i] == 0) and y1 >= 275 and y1 <= 285:
                speed[i] = estimateSpeed([x1, y1, w1, h1], [x2, y2, w2, h2])

            if speed[i] != None and y1 >= 180:
                #cv2.putText(resultImage, str(int(speed[i])) + " km/hr", (int(x1 + w1), int(y1)),cv2.FONT_HERSHEY_SIMPLEX, 0.75, (255, 255, 255), 2)
                cl = np.array(speed[i])
                il = image.shape
                w = il[0]
                h = il[1]
                tid = np.array(i)
                BB = [[y1/h, x1/w, (y1+h1)/h, (x1+w1)/w]]
                BB = np.asarray(BB)
                vis_util.visualize_boxes_and_labels_on_image_array(
                    resultImage,
                    BB,
                    cl, None,
                    category_index=None,
                    instance_masks = None,
                    use_normalized_coordinates=True,
                    line_thickness=12)

                one = [x1, y1, w1, h1]
                two = [x2, y2, w2, h2]
                print ('BuildingID ' + str(i) + ' Location1: ' + str(one) + ' Location2: ' + str(two) + ' speed is ' + str("%.2f" % round(speed[i], 0)) + ' km/h.\n')
                cv2.imshow(resultImage)

if cv2.waitKey(33) == 27:
    break

cv2.destroyAllWindows()

```

Main function call:

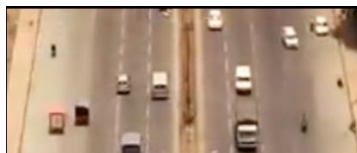
```

if __name__ == '__main__':
    video = cv2.VideoCapture(compressed_path)
    trackMultipleObjects()

```

c. Speed Estimation

Calculating the pixel width of the road: 413 px



Speed estimation function


```
[29] def estimateSpeed(location1, location2):
    d_pixels = math.sqrt(math.pow(location2[0] - location1[0], 2) + math.pow(location2[1] - location1[1], 2))
    # ppm = location2[2] / carWidtht
    ppm = 413/60;
    d_meters = d_pixels / ppm
    #print("d_pixels=" + str(d_pixels), "d_meters=" + str(d_meters))
    speed = d_meters * fps * 3.6
    return speed
```

7. RESULTS AND COMPARISON:


The screenshot shows a Google Colab notebook interface. The top part displays the code for the `estimateSpeed` function. Below the code, the notebook output shows two frames from a video. The first frame shows a building with a bounding box and the text: "BuildingID 80 Location1: [477, 284, 98, 150] Location2: [477, 284, 99, 151] speed is 0.00 km/h." The second frame shows the same building with a bounding box and the text: "BuildingID 80 Location1: [477, 284, 99, 151] Location2: [474, 283, 98, 150] speed is 50.00 km/h." The bottom of the notebook shows the execution status: "Executing (4m 15s) Cell > trackMultipleObjects() > cv2.imshow() > display() > format() > __call__() > catch_format_error() > __call__() > repr_png() > save() > _save() > _save()".

myDesk PROD Meet - wid-rend-smi Machine_Learning_Mini_Projec python - how play mp4 video in go


colab.research.google.com/drive/1_yfXsUEHIZ8PFtctzhOTwN_UeQjuAbjg#scrollTo=Puy0NattSr_F

Machine_Learning_Mini_Project.ipynb ☆
File Edit View Insert Runtime Tools Help Saving_

+ Code + Text



BuildingID 80 Location1: [477, 284, 99, 151] Location2: [474, 283, 98, 150] speed is 50.00 km/h.



Executing (4m 27s) Cell > trackMultipleObjects() > cv2.imshow() > display() > format() > __call__() > catch_format_error() > __call__() > _repr_png_() > save() > _save() > _save()


myDesk PROD Meet - wid-rend-smi CO Machine_Learning_Mini_Projec python - how play mp4 video in go +

colab.research.google.com/drive/1_yfXsUEHiZ8PFictzhOTwN_UeQjuAbjg#scrollTo=Puy0NattSr_F

Machine_Learning_Mini_Project.ipynb ☆

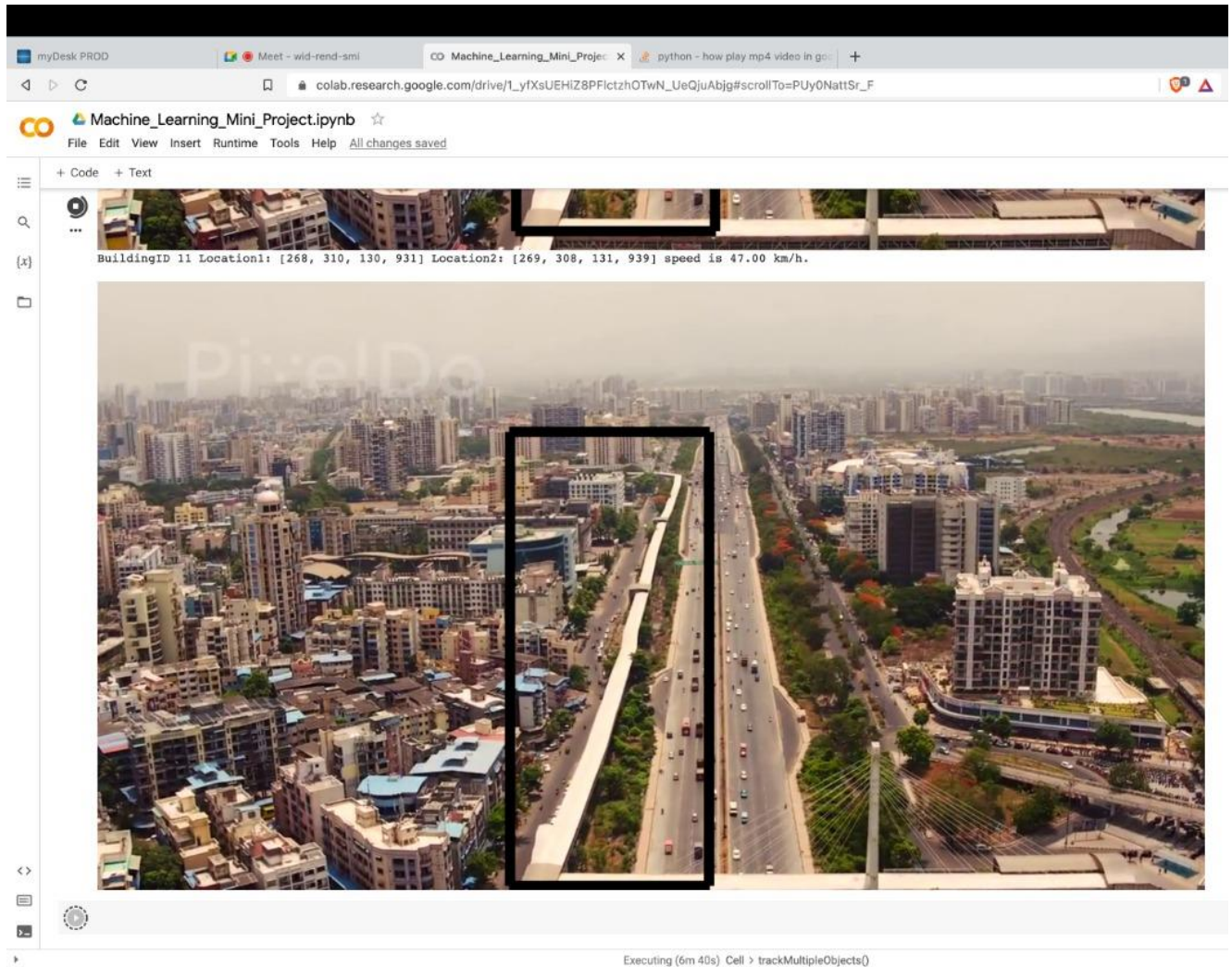
File Edit View Insert Runtime Tools Help Saving...

+ Code + Text



BuildingID 11 Location1: [276, 304, 130, 931] Location2: [272, 306, 130, 928] speed is 47.00 km/h.

Executing (5m 52s) Cell > trackMultipleObjects()



Here, the speed of the camera is estimated to be around 47-50 km/hr

8. CONCLUSION AND FUTURE ENHANCEMENTS:

This project is used to estimate the speeds of the camera from the aerial footage of drones. The speed estimation is done by tracking several objects simultaneously and thereby producing different speed estimates for the same sequence of frames. This inconsistency arises due to imprecise object detection and the differences in the camera orientation. Also improving the accuracy of the tracker helps in the consistency of the speed estimation with respect to different objects in the same frame set.

9. REFERENCES:

1. Sun, C., Zhan, W., She, J. and Zhang, Y., 2020. Object detection from the video taken by drone via convolutional neural networks. Mathematical Problems in Engineering, 2020.
2. Singh, Upasna, Anjali Saini, and Ravi Domala. "Object Tracking in Videos Using CNN." ICDSMLA 2019. Springer, Singapore, 2020. 520-527.
3. J. Wang, "Research of vehicle speed detection algorithm in video surveillance," 2016 International Conference on Audio, Language and Image Processing (ICALIP), 2016, pp. 349-352, doi: 10.1109/ICALIP.2016.7846573.

10. APPENDIX A:

The undersigned acknowledge they have completed implementing the project “Estimation of the Speed of Drones from Aerial Footage” and agree with the approach it presents.

Signature:



Date:

14/06/2022

Name:

Shreya Ananth

Signature:



Date:

14/06/2022

Name:

Sneha S