Project Report – CS436: Computer Vision

By Taha Sarwat (25100287) & Muhammad Khizer Sadiq (24100309)

Task 1:

The opening task of this assignment, despite being relatively simple, still required us to do a lot of research into the world of neural networks. Creating a custom one from scratch was a mammoth task on its own and we did not want to compromise on the quality of the output.

We created the following model that was later saved as a .h5 file:

Layer (type)	Output Shape	Param #
batch_normalization_1 (BatchNormalization)	(None, 80, 160, 3)	12
Conv1 (Conv2D)	(None, 78, 158, 8)	224
Conv2 (Conv2D)	(None, 76, 156, 16)	1168
max_pooling2d_1 (MaxPooling2D)	(None, 38, 78, 16)	0
Conv3 (Conv2D)	(None, 36, 76, 16)	2320
dropout_1 (Dropout)	(None, 36, 76, 16)	0
Conv4 (Conv2D)	(None, 34, 74, 32)	4640
dropout_2 (Dropout)	(None, 34, 74, 32)	0
Conv5 (Conv2D)	(None, 32, 72, 32)	9248
dropout_3 (Dropout)	(None, 32, 72, 32)	0
max_pooling2d_2 (MaxPooling2D)	(None, 16, 36, 32)	0
Conv6 (Conv2D)	(None, 14, 34, 64)	18496

dropout_4 (Dropout)	(None, 14, 34, 64)	0
Conv7 (Conv2D)	(None, 12, 32, 64)	36928
dropout_5 (Dropout)	(None, 12, 32, 64)	0
max_pooling2d_3 (MaxPooling2D)	(None, 6, 16, 64)	0
up_sampling2d_1 (UpSampling2D)	(None, 12, 32, 64)	0
Deconv1 (Conv2DTranspose)	(None, 14, 34, 64)	36928
dropout_6 (Dropout)	(None, 14, 34, 64)	0
Deconv2 (Conv2DTranspose)	(None, 16, 36, 64)	36928
dropout_7 (Dropout)	(None, 16, 36, 64)	0
up_sampling2d_2 (UpSampling2D)	(None, 32, 72, 64)	0
Deconv3 (Conv2DTranspose)	(None, 34, 74, 32)	18464
dropout_8 (Dropout)	(None, 34, 74, 32)	0
Deconv4 (Conv2DTranspose)	(None, 36, 76, 32)	9248
dropout_9 (Dropout)	(None, 36, 76, 32)	0

```
dropout_9 (Dropout)          (None, 36, 76, 32)        0

Deconv5 (Conv2DTranspose)    (None, 38, 78, 16)        4624

dropout_10 (Dropout)         (None, 38, 78, 16)        0

up_sampling2d_3 (UpSamplin   (None, 76, 156, 16)       0
g2D)

Deconv6 (Conv2DTranspose)    (None, 78, 158, 16)       2320

Final (Conv2DTranspose)      (None, 80, 160, 1)        145

=================================================================
Total params: 181693 (709.74 KB)
Trainable params: 181687 (709.71 KB)
Non-trainable params: 6 (24.00 Byte)
```

This neural network, which was designed in TensorFlow gave us excellent results when we ran it on **both** our video and the one presented to us in the **comma2k dataset**.

As you can observe in the following images, the result is as good as can be.


This is the same as what was given to us in the handout:

Task 1: Ground Segmentation

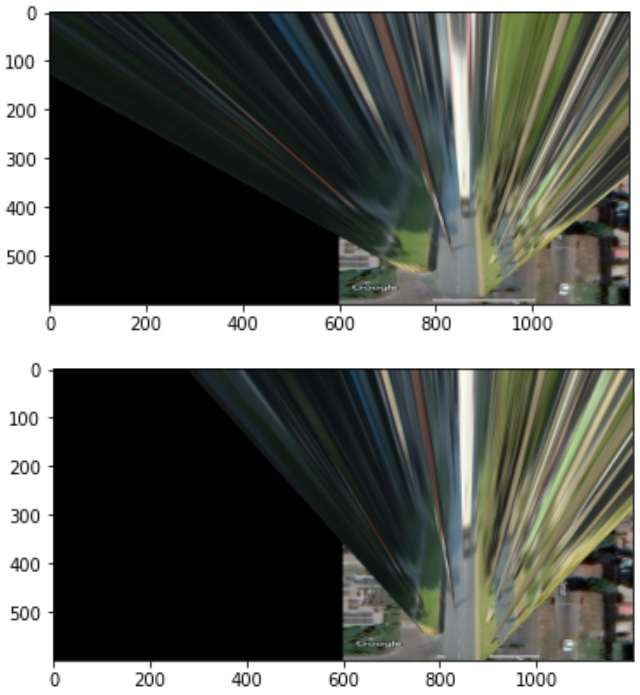
1. Apply any segmentation algorithm to segment the ground/lanes

Task 1 was well and truly, complete!

Task 2: This task involved using homography to find an aerial view of our image which was predominantly that of ground view. In order to achieve this, we had to allude to various techniques including but not limited to:

a) Normalizing image: This was done for the preprocessing part.
b) Finding *homography matrix*.
c) Using it to get the homography *array* for which we created a NumPy array.
d) Calculating SVD
e) Stitching the images together

Many of the results were staggering to say the least. Getting something so real, using a bunch of mathematical solutions was incredible to say the least. Here is a demonstration of the level of excellence we achieved in this:



Incredible images achieved!

Task 3:

a) Generating Top-View for Each Frame

This task was the most time-consuming and required us to use our problem-solving skill set. Bear in mind we used the code from **task 2** as a foundation for creating and running this code.

The first thing we needed was to create frame by frame images of the video at hand which we achieved using the following code:

```
+ Code  + Markdown | Run All  Clear All Outputs | Outline ···  Select Kernel

video_path = 'attempt1.mp4'
output_folder = "Input Images"
cap = cv2.VideoCapture(video_path)

fps = int(cap.get(cv2.CAP_PROP_FPS))

width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))

video_writer = cv2.VideoWriter("output_video.mp4", cv2.VideoWriter_fourcc(*"mp4v"), fps, (width, height))

frame_count = 0

while cap.isOpened():
    ret, frame = cap.read()
    if ret:
        frame_count += 1
        if frame_count % 10 == 0:
            cv2.imwrite(output_folder + "/" + str(frame_count) + ".jpg", frame)
            frame_count += 1
        video_writer.write(frame)
    else:
        break

frame_path = f"{output_folder}/frame_{frame_count}.jpg"
cv2.imwrite(frame_path, frame)

[56]                                                                    Python
```

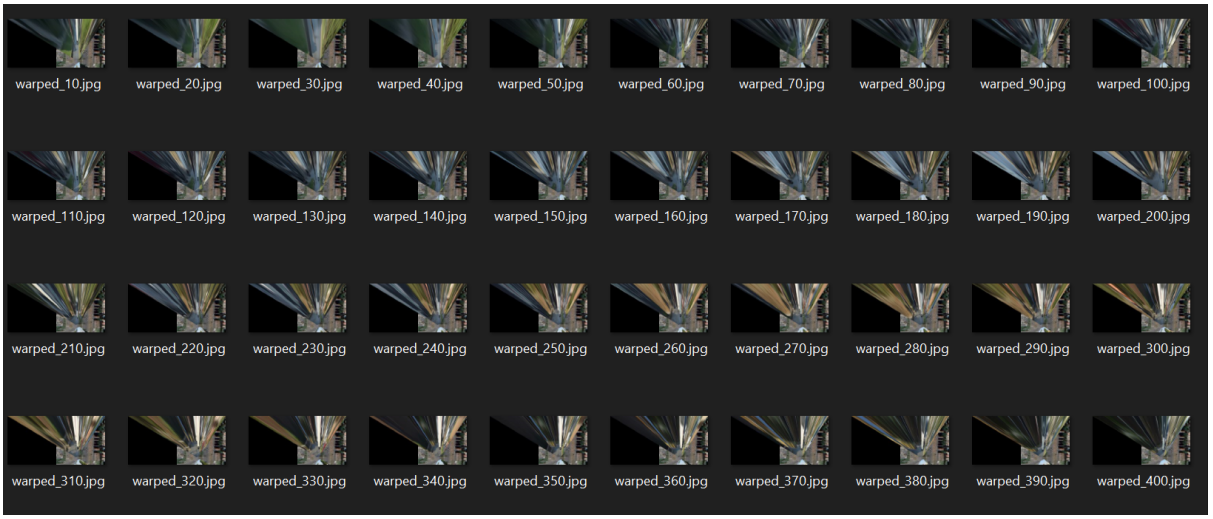
This unique and multipurpose code got us what we needed in a matter of seconds.

Later, we ran an efficient loop through our frame-by-frame images and ended up with a folder of our images:

```
+ Code  + Markdown | Run All  Clear All Outputs | Outline ···  Select Kerne

input_folder = "Input Images"
output_folder = "Output Images"
if not os.path.exists(output_folder):
    os.makedirs(output_folder)

for filename in os.listdir(input_folder):
    if filename.endswith(".jpg"):
        image_path = os.path.join(input_folder, filename)
        img1 = cv2.imread(image_path)
        img2 = cv2.imread('aerial.jpg')
        img1 = normalize_image(img1)
        img2 = normalize_image(img2)


        row1, col1 = img1.shape[:2]
        row2, col2 = img2.shape[:2]
        canvas = np.zeros((max(row1, row2), col1 + col2, 3), dtype='uint8')

        canvas[:row1, :col1, :] = img1
        canvas[:row2, col1:col1 + col2, :] = img2
        plt.imshow(canvas)
        img2_points[:,0] = img2_points[:,0] + col1

        warped_img = cv2.warpPerspective(img1, homography_matrix, (canvas.shape[1], canvas.shape[0]))
        warped_img[:, img1.shape[1]:, :] = np.where(warped_img[:, img1.shape[1]:, :] == 0, img2, warped_img[:, img1.sha
        output_path = os.path.join(output_folder, f"warped_{filename}")
        cv2.imwrite(output_path, warped_img)
```

At this point, we did not just manage to write new code to cater to the task's requirements, but also moulded the already existing code to our requirements!

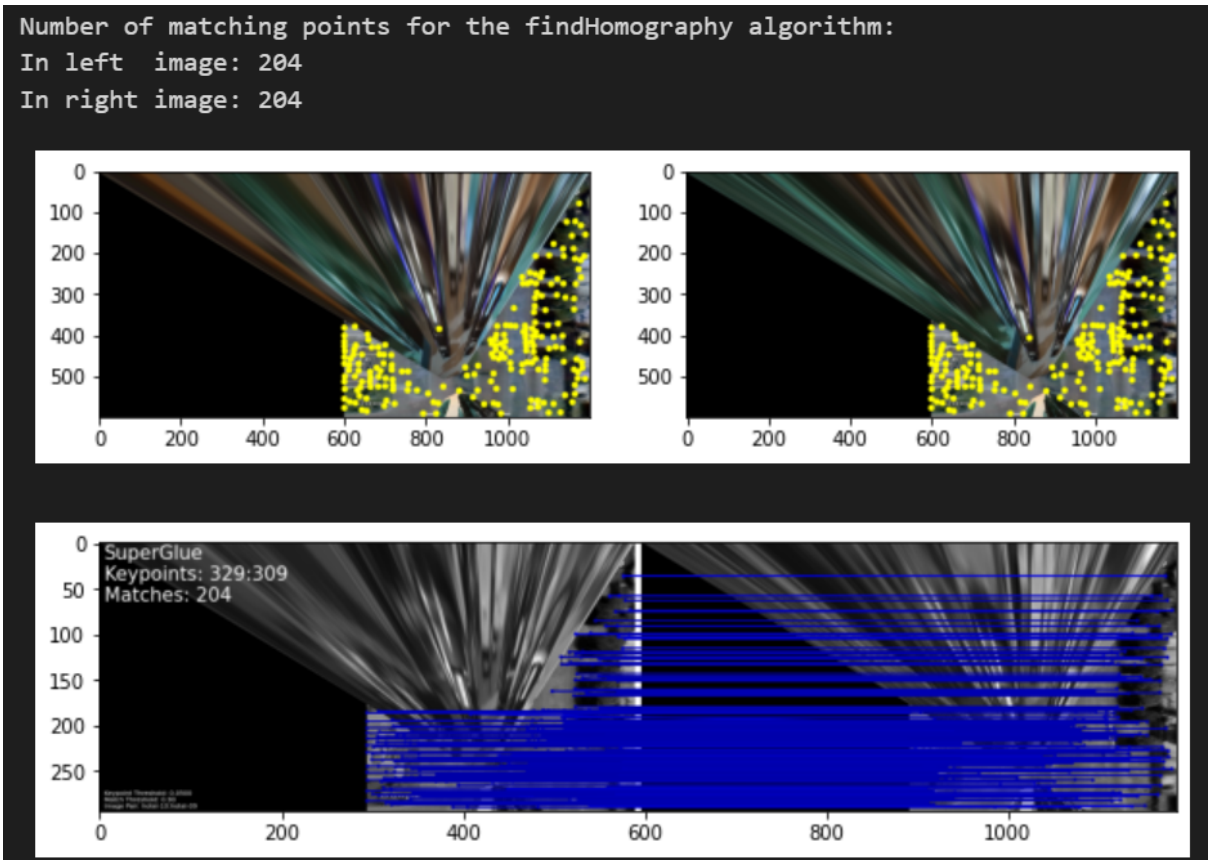
The following is an image of the folder we ended up with:

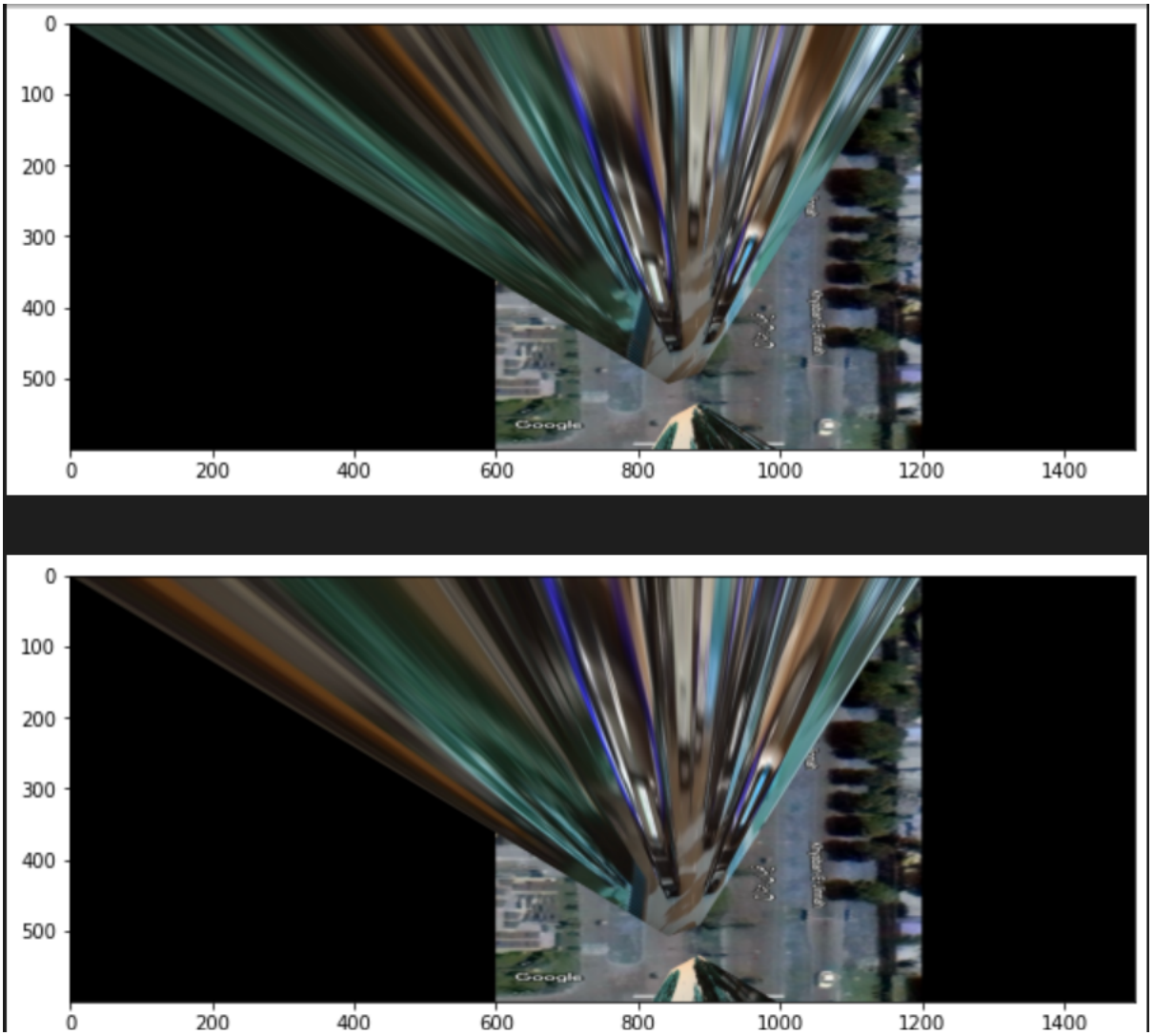


b) Stitching:

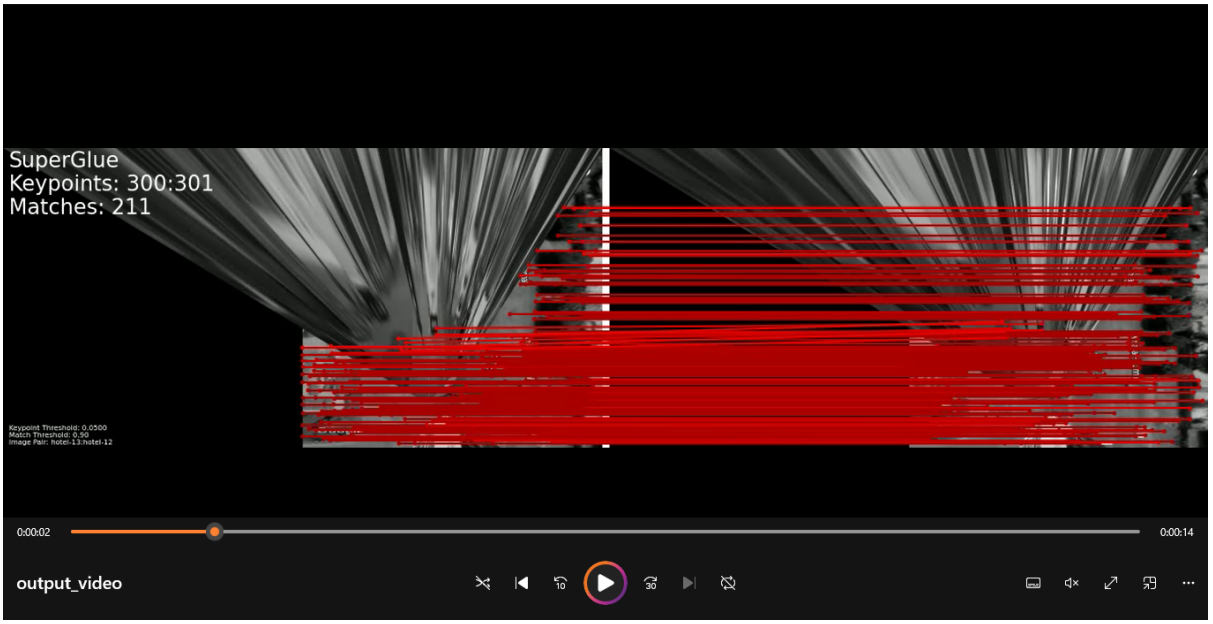
This part also required a lot of work in not just understanding the code in SuperGlue, but also editing it to run on our frames and then stitch them to generate a top view of all the frames and we achieved that emphatically!

These are just a few examples of images being merged effectively:





This is just a brief example of it working for our images. You can see how each of these are rich in detail and texture, just what we needed from this stitching tool!



Task 4:

This is where the idea of object detection came into play and ensuring ours was done effectively was an uphill task!

We found the YOLO-NAS starter pack, by dec.ai:

YOLO-NAS Starter Notebook

Deep learning firm Deci.ai has recently launched YOLO-NAS. This deep learning model delivers superior real-time object detection capabilities and high performance ready for production. These models were constructed using Deci's proprietary AutoNAC™ NAS technology. YOLO-NAS is a new real-time state-of-the-art object detection model that outperforms YOLOv7, YOLOv8 & the recently released YOLOv6 3.0 models in terms of mAP and inference latency.

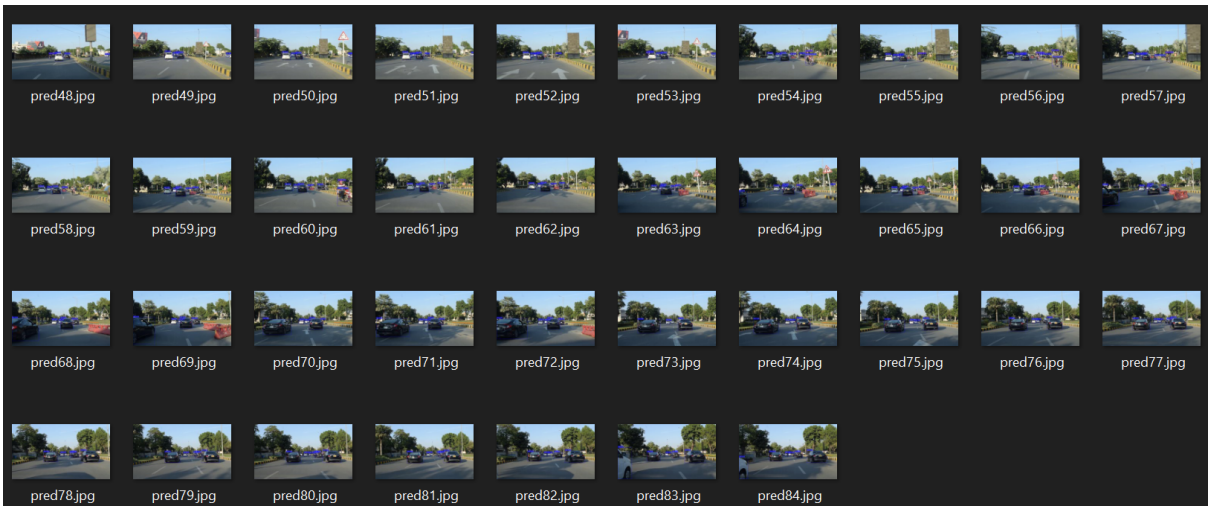
We used this to get all images we needed:

```
from PIL import Image

input_folder = '/content/drive/MyDrive/Colab Notebooks/Input Images'
output_folder = '/content/drive/MyDrive/Colab Notebooks/Output Images'

counter = 0

for filename in os.listdir(input_folder):
  if filename.endswith(('.jpg', '.jpeg', '.png')):
    input_path = os.path.join(input_folder, filename)
    output_filename = f'pred{counter}'
    output_path = os.path.join(output_folder, output_filename)

    out = model.predict(input_path, conf=0.6)

    out.save(output_path)
    counter += 1
print("Prediction and saving complete.")
```

```
[2023-12-03 14:02:34] INFO - pipelines.py - Fusing some of the model's layers. If this takes too much memory, you can deactivate it by setting `fuse_model=False`
[2023-12-03 14:02:34] INFO - pipelines.py - Fusing some of the model's layers. If this takes too much memory, you can deactivate it by setting `fuse_model=False`
[2023-12-03 14:02:35] INFO - pipelines.py - Fusing some of the model's layers. If this takes too much memory, you can deactivate it by setting `fuse_model=False`
[2023-12-03 14:02:35] INFO - pipelines.py - Fusing some of the model's layers. If this takes too much memory, you can deactivate it by setting `fuse_model=False`
[2023-12-03 14:02:35] INFO - pipelines.py - Fusing some of the model's layers. If this takes too much memory, you can deactivate it by setting `fuse_model=False`
[2023-12-03 14:02:36] INFO - pipelines.py - Fusing some of the model's layers. If this takes too much memory, you can deactivate it by setting `fuse_model=False`
[2023-12-03 14:02:36] INFO - pipelines.py - Fusing some of the model's layers. If this takes too much memory, you can deactivate it by setting `fuse_model=False`
```

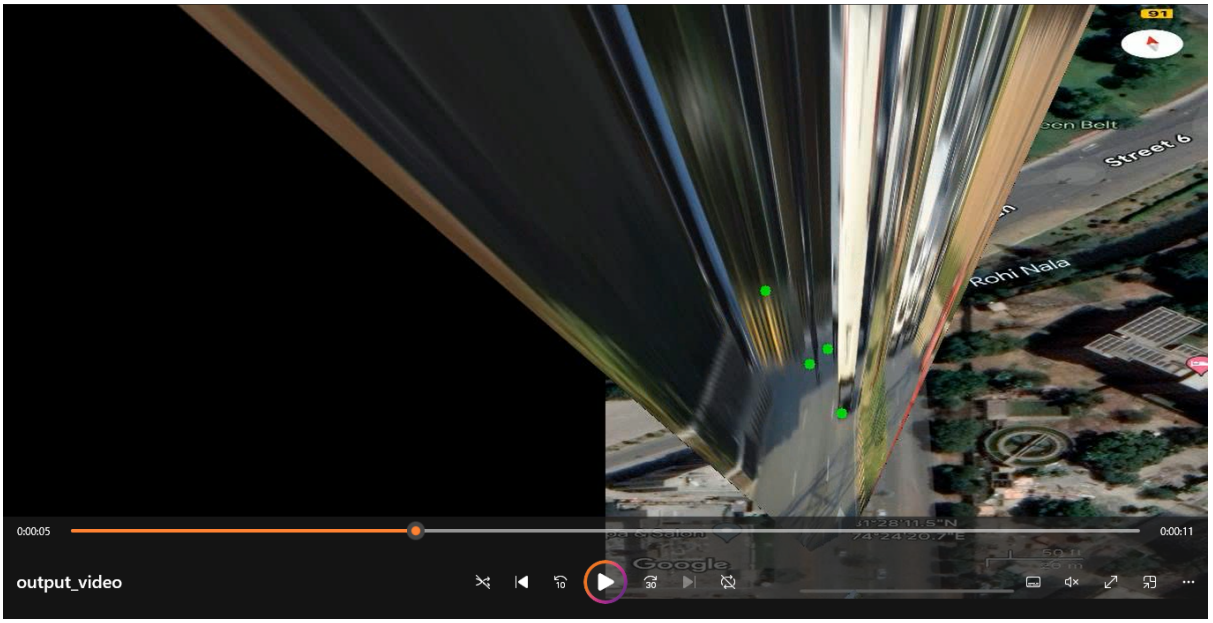
It did not take us too long, but again we would like to emphasize upon the way we used and edited our code to do it.

The results were impeccable:



We then proceeded to using these spots and projected them in aerial view as follows:

However, it is essential to mention that for the homography matrix, we had to take new points on each and every frame. Then we needed to make a video showing all the object detected warped images on top view. Here is a screenshot of the video. So the work that we put into this should not go unnoticed!



Task 5:

For this task, we had to overlay our top view on Google Earth Pro software and we had to align it well with the respective GPS coordinates. Since it was not properly overlaying, we decided to calculate the pixel projection error. The error we found by finding the GPS coordinates of the object in the original video and also the GPS coordinates from the object on the top view frame, and we got it around 31%. The overlaid image we got is:

We kind of less darkened the black box in the image above so that it remains visible along with our top view aligning it perfectly on the road with the respective coordinates.