```
# Install the necessary packages
!pip install ultralytics opencv-python-headless
```

```
Requirement already satisfied: ultralytics in /usr/local/lib/python3.12/dist-pa
Requirement already satisfied: opencv-python-headless in /usr/local/lib/python3
Requirement already satisfied: numpy>=1.23.0 in /usr/local/lib/python3.12/dist-
Requirement already satisfied: matplotlib>=3.3.0 in /usr/local/lib/python3.12/d
Requirement already satisfied: opencv-python>=4.6.0 in /usr/local/lib/python3.1
Requirement already satisfied: pillow>=7.1.2 in /usr/local/lib/python3.12/dist-
Requirement already satisfied: pyyaml>=5.3.1 in /usr/local/lib/python3.12/dist-
Requirement already satisfied: requests>=2.23.0 in /usr/local/lib/python3.12/di
Requirement already satisfied: scipy>=1.4.1 in /usr/local/lib/python3.12/dist-p
Requirement already satisfied: torch>=1.8.0 in /usr/local/lib/python3.12/dist-p
Requirement already satisfied: torchvision>=0.9.0 in /usr/local/lib/python3.12/
Requirement already satisfied: psutil in /usr/local/lib/python3.12/dist-package
Requirement already satisfied: polars in /usr/local/lib/python3.12/dist-package
Requirement already satisfied: ultralytics-thop>=2.0.18 in /usr/local/lib/pytho
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.12/di
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.12/dist-p
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.12/d
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.12/d
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.12/dis
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.12/di
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.1
Requirement already satisfied: charset_normalizer<4,>=2 in /usr/local/lib/pytho
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.12/dist-p
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.12/
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.12/
Requirement already satisfied: filelock in /usr/local/lib/python3.12/dist-packa
Requirement already satisfied: typing-extensions>=4.10.0 in /usr/local/lib/pyth
Requirement already satisfied: setuptools in /usr/local/lib/python3.12/dist-pac
Requirement already satisfied: sympy>=1.13.3 in /usr/local/lib/python3.12/dist-
Requirement already satisfied: networkx in /usr/local/lib/python3.12/dist-packa
Requirement already satisfied: jinja2 in /usr/local/lib/python3.12/dist-package
Requirement already satisfied: fsspec in /usr/local/lib/python3.12/dist-package
Requirement already satisfied: nvidia-cuda-nvrtc-cu12==12.6.77 in /usr/local/li
Requirement already satisfied: nvidia-cuda-runtime-cu12==12.6.77 in /usr/local/
Requirement already satisfied: nvidia-cuda-cupti-cu12==12.6.80 in /usr/local/li
Requirement already satisfied: nvidia-cudnn-cu12==9.10.2.21 in /usr/local/lib/p
Requirement already satisfied: nvidia-cublas-cu12==12.6.4.1 in /usr/local/lib/p
Requirement already satisfied: nvidia-cufft-cu12==11.3.0.4 in /usr/local/lib/py
Requirement already satisfied: nvidia-curand-cu12==10.3.7.77 in /usr/local/lib/
Requirement already satisfied: nvidia-cusolver-cu12==11.7.1.2 in /usr/local/lib
Requirement already satisfied: nvidia-cusparse-cu12==12.5.4.2 in /usr/local/lib
Requirement already satisfied: nvidia-cusparselt-cu12==0.7.1 in /usr/local/lib/
Requirement already satisfied: nvidia-nccl-cu12==2.27.3 in /usr/local/lib/pytho
Requirement already satisfied: nvidia-nvtx-cu12==12.6.77 in /usr/local/lib/pyth
Requirement already satisfied: nvidia-nvjitlink-cu12==12.6.85 in /usr/local/lib
Requirement already satisfied: nvidia-cufile-cu12==1.11.1.6 in /usr/local/lib/p
Requirement already satisfied: triton==3.4.0 in /usr/local/lib/python3.12/dist-
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.12/dist-packa
Requirement already satisfied: mpmath<1.4,>=1.1.0 in /usr/local/lib/python3.12/
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.12/dis
```

```
import cv2
from ultralytics import YOLO
from google.colab import files
import os

# Define the path to your uploaded video
# (This assumes 'My Movie.mp4' is in the main Colab directory)
```

```
INPUT_VIDEO_PATH = 'My Movie.mp4'
OUTPUT_VIDEO_PATH = 'output_video.mp4'
```

```
# This cell will prompt you to upload your video
print(f"Please upload your video file and name it: {INPUT_VIDEO_PATH}")
uploaded = files.upload()

uploaded_file_name = None
if INPUT_VIDEO_PATH in uploaded:
    uploaded_file_name = INPUT_VIDEO_PATH
elif len(uploaded) > 0:
    # If the exact name isn't found, but a file was uploaded, assume it's the
    uploaded_file_name = list(uploaded.keys())[0]

if uploaded_file_name:
    if uploaded_file_name != INPUT_VIDEO_PATH:
        # Rename the uploaded file to the expected name
        os.rename(uploaded_file_name, INPUT_VIDEO_PATH)
        print(f"File '{uploaded_file_name}' renamed to '{INPUT_VIDEO_PATH}'")
    print(f"\nSuccessfully uploaded '{INPUT_VIDEO_PATH}'")
else:
    print(f"\nError: No file was uploaded.")
    print("Please upload your video file and name it 'My Movie.mp4'.")
```

```
Please upload your video file and name it: My Movie.mp4
Choose files My Movie.mp4
My Movie.mp4(video/mp4) - 29331003 bytes, last modified: 10/11/2025 - 100% done
Saving My Movie.mp4 to My Movie.mp4

Successfully uploaded 'My Movie.mp4'
```

```
# 1. Load the pre-trained YOLOv8 model (yolov8n.pt is small and fast)
model = YOLO('yolov8n.pt')

# 2. Open the input video file
cap = cv2.VideoCapture(INPUT_VIDEO_PATH)

if not cap.isOpened():
    print(f"Error: Could not open video file {INPUT_VIDEO_PATH}")
else:
    # 3. Get video properties (width, height, frames-per-second)
    frame_width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
    frame_height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
    fps = int(cap.get(cv2.CAP_PROP_FPS))

    # 4. Define the video writer to save the output
    #    We use 'mp4v' codec for an .mp4 file
    fourcc = cv2.VideoWriter_fourcc(*'mp4v')
    out = cv2.VideoWriter(OUTPUT_VIDEO_PATH, fourcc, fps, (frame_width, frame_

    print("Processing video... This may take a few minutes.")

    frame_count = 0
    # 5. Loop through every frame in the video
    while cap.isOpened():
        ret, frame = cap.read()
        if not ret:
            break # End of video
        # 6. Run YOLOv8 detection on the frame
        #    'stream=True' is more efficient for video
        #    'verbose=False' hides the extra print-outs
        results = model(frame, stream=True, verbose=False)

        # 7. Get the annotated frame
        #    The '.plot()' method automatically draws all boxes, labels, and s
        for r in results:
            annotated_frame = r.plot()

            # 8. Write the annotated frame to the output video file
            out.write(annotated_frame)

        frame_count += 1
        if frame_count % 100 == 0:
            print(f"Processed {frame_count} frames...")

    # 9. Release everything
    cap.release()
    out.release()
    cv2.destroyAllWindows()

    print(f"--- Processing Complete! ---")
    print(f"Output video saved as: {OUTPUT_VIDEO_PATH}")
```

```
Downloading https://github.com/ultralytics/assets/releases/download/v8.3.0/yolo
Processing video... This may take a few minutes.
Processed 100 frames...
Processed 200 frames...
Processed 300 frames...
Processed 400 frames...
Processed 500 frames...
--- Processing Complete! ---
Output video saved as: output_video.mp4
```

```
# Download the processed video to your local computer
files.download(OUTPUT_VIDEO_PATH)
```

```
# Install the new packages required for MiDaS
!pip install transformers timm
```

```
Requirement already satisfied: transformers in /usr/local/lib/python3.12/dist-p
Requirement already satisfied: timm in /usr/local/lib/python3.12/dist-packages
Requirement already satisfied: filelock in /usr/local/lib/python3.12/dist-packa
Requirement already satisfied: huggingface-hub<1.0,>=0.34.0 in /usr/local/lib/p
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.12/dist-pa
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.12/dis
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.12/dist-pa
Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib/python3.12/d
Requirement already satisfied: requests in /usr/local/lib/python3.12/dist-packa
Requirement already satisfied: tokenizers<=0.23.0,>=0.22.0 in /usr/local/lib/py
Requirement already satisfied: safetensors>=0.4.3 in /usr/local/lib/python3.12/
Requirement already satisfied: tqdm>=4.27 in /usr/local/lib/python3.12/dist-pac
Requirement already satisfied: torch in /usr/local/lib/python3.12/dist-packages
Requirement already satisfied: torchvision in /usr/local/lib/python3.12/dist-pa
Requirement already satisfied: fsspec>=2023.5.0 in /usr/local/lib/python3.12/di
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/pyt
Requirement already satisfied: hf-xet<2.0.0,>=1.1.3 in /usr/local/lib/python3.1
Requirement already satisfied: charset_normalizer<4,>=2 in /usr/local/lib/pytho
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.12/dist-p
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.12/
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.12/
Requirement already satisfied: setuptools in /usr/local/lib/python3.12/dist-pac
Requirement already satisfied: sympy>=1.13.3 in /usr/local/lib/python3.12/dist-
Requirement already satisfied: networkx in /usr/local/lib/python3.12/dist-packa
Requirement already satisfied: jinja2 in /usr/local/lib/python3.12/dist-package
Requirement already satisfied: nvidia-cuda-nvrtc-cu12==12.6.77 in /usr/local/li
Requirement already satisfied: nvidia-cuda-runtime-cu12==12.6.77 in /usr/local/
Requirement already satisfied: nvidia-cuda-cupti-cu12==12.6.80 in /usr/local/li
Requirement already satisfied: nvidia-cudnn-cu12==9.10.2.21 in /usr/local/lib/p
Requirement already satisfied: nvidia-cublas-cu12==12.6.4.1 in /usr/local/lib/p
Requirement already satisfied: nvidia-cufft-cu12==11.3.0.4 in /usr/local/lib/py
Requirement already satisfied: nvidia-curand-cu12==10.3.7.77 in /usr/local/lib/
Requirement already satisfied: nvidia-cusolver-cu12==11.7.1.2 in /usr/local/lib
Requirement already satisfied: nvidia-cusparse-cu12==12.5.4.2 in /usr/local/lib
Requirement already satisfied: nvidia-cusparselt-cu12==0.7.1 in /usr/local/lib/
Requirement already satisfied: nvidia-nccl-cu12==2.27.3 in /usr/local/lib/pytho
Requirement already satisfied: nvidia-nvtx-cu12==12.6.77 in /usr/local/lib/pyth
Requirement already satisfied: nvidia-nvjitlink-cu12==12.6.85 in /usr/local/lib
Requirement already satisfied: nvidia-cufile-cu12==1.11.1.6 in /usr/local/lib/p
Requirement already satisfied: triton==3.4.0 in /usr/local/lib/python3.12/dist-
Requirement already satisfied: pillow!=8.3.*,>=5.3.0 in /usr/local/lib/python3.
Requirement already satisfied: mpmath<1.4,>=1.1.0 in /usr/local/lib/python3.12/
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.12/dis
```

```
import cv2
from ultralytics import YOLO
from google.colab import files
import torch
from transformers import DPTForDepthEstimation, DPTImageProcessor
import numpy as np
from PIL import Image

# --- Setup Device (use GPU if available) ---
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(f"Using device: {device}")

# --- Load YOLOv8 Model (as before) ---
model_yolo = YOLO('yolov8n.pt')

# --- Load MiDaS Depth Estimation Model ---
print("Loading MiDaS model...")
# We use the 'transformers' library to load DPT (a state-of-the-art model)
processor_midas = DPTImageProcessor.from_pretrained("Intel/dpt-large")
model_midas = DPTForDepthEstimation.from_pretrained("Intel/dpt-large")
model_midas.to(device)
print("MiDaS model loaded.")

# --- Define Video Paths ---
INPUT_VIDEO_PATH = 'My Movie.mp4'
OUTPUT_VIDEO_PATH = 'output_video_with_depth.mp4'
```

```
Using device: cuda
Loading MiDaS model...
/usr/local/lib/python3.12/dist-packages/huggingface_hub/utils/_auth.py:94: User
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access pub
  warnings.warn(
Some weights of DPTForDepthEstimation were not initialized from the model check
```

```
You should probably TRAIN this model on a down-stream task to be able to use it
MiDaS model loaded.
```

```
# Open the input video file
cap = cv2.VideoCapture(INPUT_VIDEO_PATH)

if not cap.isOpened():
    print(f"Error: Could not open video file {INPUT_VIDEO_PATH}")
else:
    # Get video properties
    frame_width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
    frame_height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
    fps = int(cap.get(cv2.CAP_PROP_FPS))

    # Define the video writer
    fourcc = cv2.VideoWriter_fourcc(*'mp4v')
    out = cv2.VideoWriter(OUTPUT_VIDEO_PATH, fourcc, fps, (frame_width, frame_

    print("Processing video with YOLO and MiDaS...")

    frame_count = 0
    while cap.isOpened():
        ret, frame = cap.read()
        if not ret:
            break

        # --- MODULE 1 (Step 2a): MiDaS Depth Estimation ---
        # Convert frame for MiDaS (OpenCV BGR to PIL RGB)
        image_pil = Image.fromarray(cv2.cvtColor(frame, cv2.COLOR_BGR2RGB))
        inputs = processor_midas(images=image_pil, return_tensors="pt").to(dev

        with torch.no_grad():
            outputs = model_midas(**inputs)
            predicted_depth = outputs.predicted_depth

        # Resize depth map to match original frame size
        prediction = torch.nn.functional.interpolate(
            predicted_depth.unsqueeze(1),
            size=image_pil.size[::-1],
            mode="bicubic",
            align_corners=False,
        ).squeeze()

        # Convert depth map to a numpy array
        depth_map_numpy = prediction.cpu().numpy()

        # Normalize depth map for better visualization (0-255)
        # You can also use the raw values for real distance calculation later
        depth_map_normalized = cv2.normalize(depth_map_numpy, None, 0, 255, cv


        # --- MODULE 1 (Step 1): YOLOv8 Object Detection ---
        # Run YOLO on the *original* frame
        results = model_yolo(frame, stream=True, verbose=False)

        # --- FUSION: Manually draw boxes and add depth text ---
        annotated_frame = frame.copy() # Start with the original frame

        for r in results:
            boxes = r.boxes
            for box in boxes:
                # 1. Get YOLO Box Coordinates
                x1, y1, x2, y2 = map(int, box.xyxy[0])

                # 2. Get Class Name and Confidence
                conf = float(box.conf[0])
                cls_id = int(box.cls[0])
                class_name = model_yolo.names[cls_id]

                # Filter for common road objects and high confidence
                if class_name in ['car', 'truck', 'bus', 'person', 'stop sign'

                    # 3. Get Depth Value
                    # Find the center of the bounding box
                    cx, cy = int((x1 + x2) / 2), int((y1 + y2) / 2)

                    # Get the depth value at the center (from the NON-normaliz
                    # We invert it because MiDaS outputs "inverse depth"
                    depth_value = 1 / depth_map_numpy[cy, cx]

                    # 4. Draw Rectangle (Green)
                    cv2.rectangle(annotated_frame, (x1, y1), (x2, y2), (0, 255

                    # 5. Create and Draw Label (White text on Green background
                    label = f"{class_name}: {depth_value:.2f}m" # "m" is a rel

                    # Get text size to create a background
                    (w, h), _ = cv2.getTextSize(label, cv2.FONT_HERSHEY_SIMPLE
                    cv2.rectangle(annotated_frame, (x1, y1 - h - 10), (x1 + w,
                    cv2.putText(annotated_frame, label, (x1, y1 - 5), cv2.FONT

        # Write the final annotated frame to the output video
        out.write(annotated_frame)

        frame_count += 1
        if frame_count % 50 == 0:
            print(f"Processed {frame_count} frames...")

    # Release everything
    cap.release()
    out.release()
    cv2.destroyAllWindows()

    print(f"--- Processing Complete! ---")
    print(f"Output video saved as: {OUTPUT_VIDEO_PATH}")
```

```
Processing video with YOLO and MiDaS...
Processed 50 frames...
Processed 100 frames...
Processed 150 frames...
Processed 200 frames...
Processed 250 frames...
Processed 300 frames...
Processed 350 frames...
Processed 400 frames...
Processed 450 frames...
Processed 500 frames...
Processed 550 frames...
--- Processing Complete! ---
Output video saved as: output_video_with_depth.mp4
```

```
# Download the processed video with depth info
files.download(OUTPUT_VIDEO_PATH)
```

```
# Install the new package for tracking
!pip install boxmot
```

```
Requirement already satisfied: numpy in /usr/local/lib/python3.12/dist-packag
Requirement already satisfied: opencv-python<5.0.0,>=4.7.0 in /usr/local/lib/p
Requirement already satisfied: pandas<3.0.0,>=2.0.0 in /usr/local/lib/python3.
Requirement already satisfied: regex<2025.0.0,>=2024.0.0 in /usr/local/lib/pyt
Requirement already satisfied: scikit-learn<2.0.0,>=1.3.0 in /usr/local/lib/py
Requirement already satisfied: torch<3.0.0,>=2.2.1 in /usr/local/lib/python3.1
Requirement already satisfied: torchvision<1.0.0,>=0.17.1 in /usr/local/lib/py
Requirement already satisfied: yacs<1.0.0,>=0.1.8 in /usr/local/lib/python3.12
Requirement already satisfied: scipy in /usr/local/lib/python3.12/dist-package
Requirement already satisfied: matplotlib in /usr/local/lib/python3.12/dist-pa
Requirement already satisfied: wcwidth in /usr/local/lib/python3.12/dist-packa
Requirement already satisfied: beautifulsoup4 in /usr/local/lib/python3.12/dis
Requirement already satisfied: filelock in /usr/local/lib/python3.12/dist-pack
Requirement already satisfied: requests[socks] in /usr/local/lib/python3.12/di
Requirement already satisfied: tqdm in /usr/local/lib/python3.12/dist-packages
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.12/dist-
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.12/dis
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.12/dist
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.
Requirement already satisfied: typing-extensions>=4.10.0 in /usr/local/lib/pyt
Requirement already satisfied: setuptools in /usr/local/lib/python3.12/dist-pa
Requirement already satisfied: sympy>=1.13.3 in /usr/local/lib/python3.12/dist
Requirement already satisfied: networkx in /usr/local/lib/python3.12/dist-pack
Requirement already satisfied: jinja2 in /usr/local/lib/python3.12/dist-packag
Requirement already satisfied: fsspec in /usr/local/lib/python3.12/dist-packag
Requirement already satisfied: nvidia-cuda-nvrtc-cu12==12.6.77 in /usr/local/l
Requirement already satisfied: nvidia-cuda-runtime-cu12==12.6.77 in /usr/local
Requirement already satisfied: nvidia-cuda-cupti-cu12==12.6.80 in /usr/local/l
Requirement already satisfied: nvidia-cudnn-cu12==9.10.2.21 in /usr/local/lib/
Requirement already satisfied: nvidia-cublas-cu12==12.6.4.1 in /usr/local/lib/
Requirement already satisfied: nvidia-cufft-cu12==11.3.0.4 in /usr/local/lib/p
Requirement already satisfied: nvidia-curand-cu12==10.3.7.77 in /usr/local/lib
Requirement already satisfied: nvidia-cusolver-cu12==11.7.1.2 in /usr/local/li
Requirement already satisfied: nvidia-cusparse-cu12==12.5.4.2 in /usr/local/li
Requirement already satisfied: nvidia-cusparselt-cu12==0.7.1 in /usr/local/lib
Requirement already satisfied: nvidia-nccl-cu12==2.27.3 in /usr/local/lib/pyth
Requirement already satisfied: nvidia-nvtx-cu12==12.6.77 in /usr/local/lib/pyt
Requirement already satisfied: nvidia-nvjitlink-cu12==12.6.85 in /usr/local/li
Requirement already satisfied: nvidia-cufile-cu12==1.11.1.6 in /usr/local/lib/
Requirement already satisfied: triton==3.4.0 in /usr/local/lib/python3.12/dist
Requirement already satisfied: pillow!=8.3.*,>=5.3.0 in /usr/local/lib/python3
Requirement already satisfied: PyYAML in /usr/local/lib/python3.12/dist-packag
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.12/dist-pack
Requirement already satisfied: mpmath<1.4,>=1.1.0 in /usr/local/lib/python3.12
Requirement already satisfied: soupsieve>1.2 in /usr/local/lib/python3.12/dist
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.12/di
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.12/d
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.12/dist-
Requirement already satisfied: PySocks!=1.5.7,>=1.5.6 in /usr/local/lib/pythor
```

```
import cv2
from ultralytics import YOLO
from google.colab import files
import torch
from transformers import DPTForDepthEstimation, DPTImageProcessor
import numpy as np
from PIL import Image
from boxmot import create_tracker # Import the factory function
import os # Added for path manipulation

# --- Setup Device (use GPU if available) ---
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(f"Using device: {device}")

# --- Load YOLOv8 Model (as before) ---
model_yolo = YOLO('yolov8n.pt')

# --- Load MiDaS Depth Estimation Model ---
print("Loading MiDaS model...")
# We use the 'transformers' library to load DPT (a state-of-the-art model)
processor_midas = DPTImageProcessor.from_pretrained("Intel/dpt-large")
model_midas = DPTForDepthEstimation.from_pretrained("Intel/dpt-large")
model_midas.to(device)
print("MiDaS model loaded.")

# --- 3. Initialize the Tracker (Using ByteTrack) ---
tracker = None # Initialize tracker to None
try:
    print("Initializing ByteTrack tracker...")
    # --- THIS IS THE FIX ---
    # We are switching from 'deepsort' (which is broken)
    # to 'bytetrack', which is included and works.
    tracker = create_tracker(
        tracker_type='bytetrack', # <-- THE FIX
        device=device,
        half=True
    )
    if tracker is not None:
        print("--- Tracker initialized SUCCESSFULLY. ---")
    else:
        print("--- Tracker initialization FAILED (returned None). ---")

except Exception as e:
    # This will catch any errors
    print(f"--- AN ERROR OCCURRED during tracker initialization: {e} ---")

# --- Define Video Paths ---
INPUT_VIDEO_PATH = 'My Movie.mp4'
OUTPUT_VIDEO_PATH = 'output_video_with_tracking.mp4'
```

```
Using device: cuda
Loading MiDaS model...
Some weights of DPTForDepthEstimation were not initialized from the model check
You should probably TRAIN this model on a down-stream task to be able to use it
2025-11-16 07:05:17.914 | MainProcess/MainThread | INFO     | /usr/local/lib/py
MiDaS model loaded.
Initializing ByteTrack tracker...
--- Tracker initialized SUCCESSFULLY. ---
2025-11-16 07:05:17.919 | MainProcess/MainThread | INFO     | /usr/local/lib/py
```

```
2025-11-16 07:05:17.920 | MainProcess/MainThread | INFO     | /usr/local/lib/py
2025-11-16 07:05:17.920 | MainProcess/MainThread | INFO     | /usr/local/lib/py
```

```
# --- MAIN LOOP FOR MODULE 1 ONLY (YOLO + MiDaS + Tracking) ---

# Open the input video file
cap = cv2.VideoCapture(INPUT_VIDEO_PATH)

if not cap.isOpened():
    print(f"Error: Could not open video file {INPUT_VIDEO_PATH}")
else:
    if tracker is None:
        print("Warning: Tracker was not initialized. Proceeding without tracki

    frame_width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
    frame_height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
    fps = int(cap.get(cv2.CAP_PROP_FPS))

    # Define the video writer
    OUTPUT_VIDEO_PATH = 'output_video_with_tracking.mp4'
    fourcc = cv2.VideoWriter_fourcc(*'mp4v')
    out = cv2.VideoWriter(OUTPUT_VIDEO_PATH, fourcc, fps, (frame_width, frame_

    print(f"--- PROCESSING: Running Module 1 (YOLO, MiDaS, ByteTrack) ---")

    frame_count = 0
    while cap.isOpened():
        ret, frame = cap.read()
        if not ret:
            break

        # Force resize frame to 1280x720 for consistency
        if frame.shape[1] != 1280 or frame.shape[0] != 720:
            frame = cv2.resize(frame, (1280, 720))

        # --- MODULE 1 (Step 2a): MiDaS Depth Estimation ---
        image_pil = Image.fromarray(cv2.cvtColor(frame, cv2.COLOR_BGR2RGB))
        inputs = processor_midas(images=image_pil, return_tensors="pt").to(dev

        with torch.no_grad():
            outputs = model_midas(**inputs)
            predicted_depth = outputs.predicted_depth

        prediction = torch.nn.functional.interpolate(
            predicted_depth.unsqueeze(1),
            size=image_pil.size[::-1],
            mode="bicubic",
            align_corners=False,
        ).squeeze()
        depth_map_numpy = prediction.cpu().numpy()

        # --- MODULE 1 (Step 1): YOLOv8 Object Detection ---
        results = model_yolo(frame, stream=True, verbose=False)

        # --- MODULE 1 (Step 3): Object Tracking ---
        detections = []
        for r in results:
            for box in r.boxes:
                detections.append(
                    [int(b) for b in box.xyxy[0]] + [float(box.conf[0]), int(b
                )
        detections_np = np.array(detections)

        # Expect 8 columns from tracker
        tracks = np.empty((0, 8))
        if tracker is not None and len(detections_np) > 0:
            # Pass single Nx6 array to update
            tracks = tracker.update(detections_np, frame)

        # --- FUSION: Draw tracked boxes with ID and Depth ---
        annotated_frame = frame.copy()

        for track in tracks:
            # Unpack 8 values
            x1, y1, x2, y2, track_id, conf, cls_id, _ = track
            x1, y1, x2, y2, track_id = map(int, [x1, y1, x2, y2, track_id])
            class_name = model_yolo.names[int(cls_id)]

            # Filter for objects we care about (and ignore the 'person' driver
            if class_name in ['car', 'truck', 'bus', 'stop sign', 'traffic lig

                # --- Get Depth Value ---
                cx, cy = int((x1 + x2) / 2), int((y1 + y2) / 2)
                if 0 <= cy < depth_map_numpy.shape[0] and 0 <= cx < depth_map_
                    # Use 1/dist for a more intuitive "meters" (relative)
                    dist = 1 / depth_map_numpy[cy, cx]
                    depth_value = dist if dist < 1000 and dist > 0 else 0
                else:
                    depth_value = 0

                # --- Draw the Box and Label ---
                label = f"ID: {track_id} {class_name} {depth_value:.2f}m"

                cv2.rectangle(annotated_frame, (x1, y1), (x2, y2), (255, 0, 0)
                (w, h), _ = cv2.getTextSize(label, cv2.FONT_HERSHEY_SIMPLEX, 0
                cv2.rectangle(annotated_frame, (x1, y1 - h - 10), (x1 + w, y1)
                cv2.putText(annotated_frame, label, (x1, y1 - 5), cv2.FONT_HER

        # Write the final annotated frame
        out.write(annotated_frame)

        frame_count += 1
        if frame_count % 50 == 0:
            print(f"Processed {frame_count} frames...")

    # Release everything
    cap.release()
    out.release()
    cv2.destroyAllWindows()

    print(f"--- Module 1 Processing Complete! ---")
    print(f"Output video saved as: {OUTPUT_VIDEO_PATH}")
```

```
--- PROCESSING: Running Module 1 (YOLO, MiDaS, ByteTrack) ---
Processed 50 frames...
Processed 100 frames...
Processed 150 frames...
Processed 200 frames...
Processed 250 frames...
Processed 300 frames...
Processed 350 frames...
Processed 400 frames...
Processed 450 frames...
Processed 500 frames...
Processed 550 frames...
--- Module 1 Processing Complete! ---
Output video saved as: output_video_with_tracking.mp4
```

```
from google.colab import files
import os

INPUT_FILE = '/content/output_video_with_tracking.mp4'
OUTPUT_FILE_FOR_MAC = '/content/output_video_with_tracking_FOR_MAC.mov'

# Check if the input file exists
if not os.path.exists(INPUT_FILE):
    print(f"--- ERROR ---")
    print(f"The input file '{INPUT_FILE}' was not found.")
    print("This means your main processing loop (the cell you just posted) faile
else:
    print(f"Converting '{INPUT_FILE}' to a Mac-compatible .mov file...")

    # Use FFmpeg to re-encode the video with the H.264 (avc1) codec
    !ffmpeg -i "{INPUT_FILE}" -c:v libx264 -pix_fmt yuv420p -f mov "{OUTPUT_FILE

    print(f"\n--- Conversion Complete! ---")
    print(f"Your new file is ready to download: {OUTPUT_FILE_FOR_MAC}")

    # Download the new file
    files.download(OUTPUT_FILE_FOR_MAC)
```

Moduke -2 planning Module

```
# We already have 'transformers' from the MiDaS step,
# but we'll run this to make sure.
!pip install transformers timm
```

```
Requirement already satisfied: transformers in /usr/local/lib/python3.12/dist-p
Requirement already satisfied: timm in /usr/local/lib/python3.12/dist-packages 
Requirement already satisfied: filelock in /usr/local/lib/python3.12/dist-packa
Requirement already satisfied: huggingface-hub<1.0,>=0.34.0 in /usr/local/lib/p
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.12/dist-pa
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.12/dis
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.12/dist-pa
Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib/python3.12/d
Requirement already satisfied: requests in /usr/local/lib/python3.12/dist-packa
Requirement already satisfied: tokenizers<=0.23.0,>=0.22.0 in /usr/local/lib/py
Requirement already satisfied: safetensors>=0.4.3 in /usr/local/lib/python3.12/
Requirement already satisfied: tqdm>=4.27 in /usr/local/lib/python3.12/dist-pac
Requirement already satisfied: torch in /usr/local/lib/python3.12/dist-packages
Requirement already satisfied: torchvision in /usr/local/lib/python3.12/dist-pa
Requirement already satisfied: fsspec>=2023.5.0 in /usr/local/lib/python3.12/di
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/pyt
Requirement already satisfied: hf-xet<2.0.0,>=1.1.3 in /usr/local/lib/python3.1
Requirement already satisfied: charset_normalizer<4,>=2 in /usr/local/lib/pytho
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.12/dist-p
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.12/
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.12/
Requirement already satisfied: setuptools in /usr/local/lib/python3.12/dist-pac
Requirement already satisfied: sympy>=1.13.3 in /usr/local/lib/python3.12/dist-
Requirement already satisfied: networkx in /usr/local/lib/python3.12/dist-packa
Requirement already satisfied: jinja2 in /usr/local/lib/python3.12/dist-package
Requirement already satisfied: nvidia-cuda-nvrtc-cu12==12.6.77 in /usr/local/li
Requirement already satisfied: nvidia-cuda-runtime-cu12==12.6.77 in /usr/local/
Requirement already satisfied: nvidia-cuda-cupti-cu12==12.6.80 in /usr/local/li
Requirement already satisfied: nvidia-cudnn-cu12==9.10.2.21 in /usr/local/lib/p
Requirement already satisfied: nvidia-cublas-cu12==12.6.4.1 in /usr/local/lib/p
Requirement already satisfied: nvidia-cufft-cu12==11.3.0.4 in /usr/local/lib/py
Requirement already satisfied: nvidia-curand-cu12==10.3.7.77 in /usr/local/lib/
Requirement already satisfied: nvidia-cusolver-cu12==11.7.1.2 in /usr/local/lib
Requirement already satisfied: nvidia-cusparse-cu12==12.5.4.2 in /usr/local/lib
Requirement already satisfied: nvidia-cusparselt-cu12==0.7.1 in /usr/local/lib/
Requirement already satisfied: nvidia-nccl-cu12==2.27.3 in /usr/local/lib/pytho
Requirement already satisfied: nvidia-nvtx-cu12==12.6.77 in /usr/local/lib/pyth
Requirement already satisfied: nvidia-nvjitlink-cu12==12.6.85 in /usr/local/lib
Requirement already satisfied: nvidia-cufile-cu12==1.11.1.6 in /usr/local/lib/p
Requirement already satisfied: triton==3.4.0 in /usr/local/lib/python3.12/dist-
Requirement already satisfied: pillow!=8.3.*,>=5.3.0 in /usr/local/lib/python3.
Requirement already satisfied: mpmath<1.4,>=1.1.0 in /usr/local/lib/python3.12/
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.12/dis
```

```
Converting '/content/output_video_with_tracking.mp4' to a Mac-compatible .mov
ffmpeg version 4.4.2-0ubuntu0.22.04.1 Copyright (c) 2000-2021 the FFmpeg deve
  built with gcc 11 (Ubuntu 11.2.0-19ubuntu1)
  configuration: --prefix=/usr --extra-version=0ubuntu0.22.04.1 --toolchain=ha
  libavutil      56. 70.100 / 56. 70.100
  libavcodec     58.134.100 / 58.134.100
  libavformat    58. 76.100 / 58. 76.100
  libavdevice    58. 13.100 / 58. 13.100
  libavfilter     7.110.100 /  7.110.100
  libswscale      5.  9.100 /  5.  9.100
  libswresample   3.  9.100 /  3.  9.100
  libpostproc    55.  9.100 / 55.  9.100
Input #0, mov,mp4,m4a,3gp,3g2,mj2, from '/content/output_video_with_tracking.r
  Metadata:
    major_brand     : isom
    minor_version   : 512
    compatible_brands: isomiso2mp41
    encoder         : Lavf59.27.100
  Duration: 00:00:19.97, start: 0.000000, bitrate: 11075 kb/s
  Stream #0:0(und): Video: mpeg4 (Simple Profile) (mp4v / 0x7634706D), yuv420(
    Metadata:
      handler_name    : VideoHandler
      vendor_id       : [0][0][0][0]
```

```python
import cv2
from ultralytics import YOLO
from google.colab import files
import torch
from transformers import DPTForDepthEstimation, DPTImageProcessor, SegformerFo
import numpy as np
from PIL import Image
from boxmot import create_tracker # Corrected import
import os
import heapq
import time # Import time for the main loop later

# --- Setup Device (use GPU if available) ---
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(f"Using device: {device}")

# --- 1. Load YOLOv8 Model ---
model_yolo = YOLO('yolov8n.pt')

# --- 2. Load MiDaS Depth Estimation Model ---
print("Loading MiDaS model...")
# The "not initialized" warning is normal
processor_midas = DPTImageProcessor.from_pretrained("Intel/dpt-large")
model_midas = DPTForDepthEstimation.from_pretrained("Intel/dpt-large")
model_midas.to(device)
print("MiDaS model loaded.")

# --- 3. Initialize the Tracker (Using ByteTrack) ---
tracker = None # Initialize tracker to None
try:
    print("Initializing ByteTrack tracker...")
    # --- THIS IS THE FIX ---
    # We are switching from 'deepsort' (which is broken)
    # to 'bytetrack', which is included and works.
    tracker = create_tracker(
        tracker_type='bytetrack', # <-- THE FIX
        device=device,
        half=True
    )
```

```
        if tracker is not None:
            print("--- Tracker initialized SUCCESSFULLY. ---")
        else:
            print("--- Tracker initialization FAILED (returned None). ---")

    except Exception as e:
        # This will catch any errors
        print(f"--- AN ERROR OCCURRED during tracker initialization: {e} ---")

    # --- 4. Load SegFormer Segmentation Model ---
    print("Loading SegFormer model for Drivable Area...")
    seg_processor = SegformerImageProcessor.from_pretrained("nvidia/segformer-b0-f
    seg_model = SegformerForSemanticSegmentation.from_pretrained("nvidia/segformer
    seg_model.to(device)
    print("SegFormer model loaded.")



    # --- Define Video Paths ---
    INPUT_VIDEO_PATH = 'My Movie.mp4'
    OUTPUT_VIDEO_PATH = 'output_video_with_ASTAR_path.mp4' # Path for the A* step
```

```
Using device: cuda
Loading MiDaS model...
Some weights of DPTForDepthEstimation were not initialized from the model check
You should probably TRAIN this model on a down-stream task to be able to use it
2025-11-16 07:09:49.660 | MainProcess/MainThread | INFO     | /usr/local/lib/py
2025-11-16 07:09:49.662 | MainProcess/MainThread | INFO     | /usr/local/lib/py
2025-11-16 07:09:49.662 | MainProcess/MainThread | INFO     | /usr/local/lib/py
2025-11-16 07:09:49.662 | MainProcess/MainThread | INFO     | /usr/local/lib/py
2025-11-16 07:09:49.663 | MainProcess/MainThread | INFO     | /usr/local/lib/py
2025-11-16 07:09:49.663 | MainProcess/MainThread | INFO     | /usr/local/lib/py
2025-11-16 07:09:49.663 | MainProcess/MainThread | INFO     | /usr/local/lib/py
2025-11-16 07:09:49.663 | MainProcess/MainThread | INFO     | /usr/local/lib/py
2025-11-16 07:09:49.663 | MainProcess/MainThread | INFO     | /usr/local/lib/py
2025-11-16 07:09:49.664 | MainProcess/MainThread | INFO     | /usr/local/lib/py
2025-11-16 07:09:49.666 | MainProcess/MainThread | SUCCESS  | /usr/local/lib/py
MiDaS model loaded.
Initializing ByteTrack tracker...
--- Tracker initialized SUCCESSFULLY. ---
Loading SegFormer model for Drivable Area...
/usr/local/lib/python3.12/dist-packages/transformers/image_processing_base.py:4
  image_processor = cls(**image_processor_dict)
SegFormer model loaded.
```

```
    # --- Define Grid Scaling ---
    GRID_SCALE_FACTOR = 16
    GRID_WIDTH = 1280 // GRID_SCALE_FACTOR
    GRID_HEIGHT = 720 // GRID_SCALE_FACTOR

    # --- Open Video ---
    cap = cv2.VideoCapture(INPUT_VIDEO_PATH)

    if not cap.isOpened():
        print(f"Error: Could not open video file {INPUT_VIDEO_PATH}")
    else:
        if tracker is None:
            print("Warning: Tracker was not initialized. Proceeding without tracki

        frame_width, frame_height = 1280, 720
        fps = int(cap.get(cv2.CAP_PROP_FPS))
```

```
# Use 'mp4v' and the correct output path for this step
fourcc = cv2.VideoWriter_fourcc(*'mp4v')
OUTPUT_VIDEO_PATH = 'output_video_with_ASTAR_path.mp4' # This is the file

out = cv2.VideoWriter(OUTPUT_VIDEO_PATH, fourcc, fps, (frame_width, frame_

print(f"--- PROCESSING: Running Modules 1 & 2 (Perception + Planning) ---"

frame_count = 0
while cap.isOpened():
    ret, frame = cap.read()
    if not ret: break

    # Force resize frame
    frame = cv2.resize(frame, (frame_width, frame_height))
    image_pil = Image.fromarray(cv2.cvtColor(frame, cv2.COLOR_BGR2RGB))
    annotated_frame = frame.copy()

    # === MODULE 1: PERCEPTION ===
    # (MiDaS, YOLO, ByteTrack)
    inputs_midas = processor_midas(images=image_pil, return_tensors="pt").
    with torch.no_grad(): outputs_midas = model_midas(**inputs_midas)
    predicted_depth = outputs_midas.predicted_depth
    prediction_midas = torch.nn.functional.interpolate(predicted_depth.uns
    depth_map_numpy = prediction_midas.cpu().numpy()

    results_yolo = model_yolo(frame, stream=True, verbose=False)

    detections = []
    for r in results_yolo:
        for box in r.boxes:
            detections.append(
                [int(b) for b in box.xyxy[0]] + [float(box.conf[0]), int(b
            )
    detections_np = np.array(detections)

    # --- FIX 1: Expect 8 columns ---
    tracks = np.empty((0, 8))
    if tracker is not None and len(detections_np) > 0:
        tracks = tracker.update(detections_np, frame)

    # === MODULE 2: PLANNING ===
    # (SegFormer, A*)
    inputs_seg = seg_processor(images=image_pil, return_tensors="pt").to(d
    with torch.no_grad(): outputs_seg = seg_model(**inputs_seg)
    logits = outputs_seg.logits
    seg_map = torch.nn.functional.interpolate(logits, size=frame.shape[:2]
    road_mask = (seg_map == 0).astype(np.uint8)

    cost_map_grid = cv2.resize(road_mask, (GRID_WIDTH, GRID_HEIGHT), inter
    cost_map_grid = 1 - cost_map_grid # 1=obstacle, 0=drivable

    if tracker is not None:
        for track in tracks:
            # --- FIX 2: Unpack 8 values ---
            x1, y1, x2, y2, track_id, conf, cls_id, _ = track

            class_name = model_yolo.names[int(cls_id)]
            # Ignore the 'person' (driver) in planning
            if class_name in ['car', 'truck', 'bus', 'cyclist']:
```

```
                        x1_g, y1_g = int(x1/GRID_SCALE_FACTOR), int(y1/GRID_SCALE_
                        x2_g, y2_g = int(x2/GRID_SCALE_FACTOR), int(y2/GRID_SCALE_
                        if 0 <= y1_g < GRID_HEIGHT and 0 <= y2_g < GRID_HEIGHT and
                            cost_map_grid[y1_g:y2_g, x1_g:x2_g] = 1

            start_node = (GRID_WIDTH // 2, GRID_HEIGHT - 1)
            goal_node = None
            for y in range(GRID_HEIGHT // 2, 0, -1):
                if cost_map_grid[y, GRID_WIDTH // 2] == 0: goal_node = (GRID_WIDTH
            if goal_node is None: goal_node = (GRID_WIDTH // 2, GRID_HEIGHT // 2)

            path = None
            if cost_map_grid[start_node[1], start_node[0]] == 0:
                path = find_path_a_star(cost_map_grid, start_node, goal_node)

            # === VISUALIZATION (Module 1 + 2) ===

            # Draw Drivable Area (Green)
            road_overlay = np.zeros_like(annotated_frame, dtype=np.uint8)
            road_overlay[road_mask == 1] = (0, 255, 0)
            annotated_frame = cv2.addWeighted(annotated_frame, 1.0, road_overlay,

            # Draw Tracked Objects (Blue)
            if tracker is not None:
                for track in tracks:
                    # --- FIX 3: Unpack 8 values ---
                    x1, y1, x2, y2, track_id, conf, cls_id, _ = track

                    x1, y1, x2, y2, track_id = map(int, [x1, y1, x2, y2, track_id]
                    class_name = model_yolo.names[int(cls_id)]

                    # Don't draw the 'person' (driver)
                    if class_name in ['car', 'truck', 'bus', 'cyclist']:

                        # --- ALL VISUALIZATION CODE IS HERE ---

                        # 1. Get Depth Value
                        cx, cy = int((x1 + x2) / 2), int((y1 + y2) / 2)
                        if 0 <= cy < depth_map_numpy.shape[0] and 0 <= cx < depth_
                            dist = 1 / depth_map_numpy[cy, cx]
                            depth_value = dist if dist < 1000 and dist > 0 else 0
                        else:
                            depth_value = 0

                        # 2. Create Label
                        label = f"ID: {track_id} {class_name} {depth_value:.2f}m"

                        # 3. Draw Box
                        cv2.rectangle(annotated_frame, (x1, y1), (x2, y2), (255, 0

                        # 4. Draw Label Background
                        (w, h), _ = cv2.getTextSize(label, cv2.FONT_HERSHEY_SIMPLE
                        cv2.rectangle(annotated_frame, (x1, y1 - h - 10), (x1 + w,

                        # 5. Draw Label Text
                        cv2.putText(annotated_frame, label, (x1, y1 - 5), cv2.FONT
                        # --- END OF VISUALIZATION CODE ---

            # Draw A* Path (Red)
            if path:
```

```
            path_scaled = (np.array(path) * GRID_SCALE_FACTOR + GRID_SCALE_FAC
            cv2.polylines(annotated_frame, [path_scaled], isClosed=False, colo

        out.write(annotated_frame)

        frame_count += 1
        if frame_count % 50 == 0: print(f"Processed {frame_count} frames...")

    # Release everything
    cap.release()
    out.release()
    cv2.destroyAllWindows()

    print(f"--- Module 1+2 Processing Complete! ---")
    print(f"Output video saved as: {OUTPUT_VIDEO_PATH}")
```

```
--- PROCESSING: Running Modules 1 & 2 (Perception + Planning) ---
Processed 50 frames...
Processed 100 frames...
Processed 150 frames...
Processed 200 frames...
Processed 250 frames...
Processed 300 frames...
Processed 350 frames...
Processed 400 frames...
Processed 450 frames...
Processed 500 frames...
Processed 550 frames...
--- Module 1+2 Processing Complete! ---
Output video saved as: output_video_with_ASTAR_path.mp4
```

```
from google.colab import files
import os

# This is the file your main loop just created
INPUT_FILE = '/content/output_video_with_ASTAR_path.mp4'

# This is the new, Mac-compatible file we will create
OUTPUT_FILE_FOR_MAC = '/content/output_video_with_ASTAR_path_FOR_MAC.mov'

# Check if the input file exists
if not os.path.exists(INPUT_FILE):
    print(f"--- ERROR ---")
    print(f"The input file '{INPUT_FILE}' was not found.")
    print("This means your main processing loop (the cell you just ran) failed
else:
    print(f"Converting '{INPUT_FILE}' to a Mac-compatible .mov file...")

    # Use FFmpeg to re-encode the video with the H.264 (avc1) codec
    !ffmpeg -i "{INPUT_FILE}" -c:v libx264 -pix_fmt yuv420p -f mov "{OUTPUT_FILE

    print(f"\n--- Conversion Complete! ---")
    print(f"Your new file is ready to download: {OUTPUT_FILE_FOR_MAC}")

    # Download the new file
    files.download(OUTPUT_FILE_FOR_MAC)
```

```
# Download the processed video with planning overlay
files.download(OUTPUT_VIDEO_PATH)
```

```
  built with gcc 11 (Ubuntu 11.2.0-19ubuntu1)
  configuration: --prefix=/usr --extra-version=0ubuntu0.22.04.1 --toolchain=ha
  libavutil      56. 70.100 / 56. 70.100
  libavcodec     58.134.100 / 58.134.100
  libavformat    58. 76.100 / 58. 76.100
  libavdevice    58. 13.100 / 58. 13.100
```

A^star Algorithm

```
# No new installs needed, but we'll re-run this to be safe
!pip install ultralytics transformers timm boxmot
```

```
  libpostproc    55.  9.100 / 55.  9.100
Requirement already satisfied: filelock in /usr/local/lib/python3.12/dist-pack
Input #0, mov,mp4,m4a,3gp,3g2,mj2, from '/content/output_video_with_ASTAR_path
Requirement already satisfied: huggingface-hub<1.0,>=0.34.0 in /usr/local/lib/
  Metadata:
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.12/di
    major_brand     : isom
Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib/python3.12/
    minor_version   : 512
Requirement already satisfied: tokenizers<=0.23.0,>=0.22.0 in /usr/local/lib/p
    compatible_brands: isomiso2mp41
Requirement already satisfied: safetensors>=0.4.3 in /usr/local/lib/python3.12
    encoder         : Lavf59.27.100
Requirement already satisfied: tqdm>=4.27 in /usr/local/lib/python3.12/dist-pa
  Duration: 00:00:19.97, start: 0.000000, bitrate: 12062 kb/s
Requirement already satisfied: click>=8.1.8 in /usr/local/lib/python3.12/dist-
  Stream #0:0(und): Video: mpeg4 (Simple Profile) (mp4v / 0x7634706D), yuv420p
Requirement already satisfied: filterpy<2.0.0,>=1.4.5 in /usr/local/lib/python
    Metadata:
Requirement already satisfied: ftfy<7.0.0,>=6.1.3 in /usr/local/lib/python3.12
      handler_name    : VideoHandler
Requirement already satisfied: gdown<6.0.0,>=5.1.0 in /usr/local/lib/python3.1
      vendor_id       : [0][0][0][0]
Requirement already satisfied: lapx<1.0.0,>=0.5.5 in /usr/local/lib/python3.12
File '/content/output_video_with_ASTAR_path_FOR_MAC.mov' already exists. Overw
Requirement already satisfied: loguru<1.0.0,>=0.7.2 in /usr/local/lib/python3.
Stream mapping:
Requirement already satisfied: pandas<3.0.0,>=2.0.0 in /usr/local/lib/python3.
  Stream #0:0 -> #0:0 (mpeg4 (native) -> h264 (libx264))
Requirement already satisfied: scikit-learn<2.0.0,>=1.3.0 in /usr/local/lib/py
Press [q] to stop, [?] for help
Requirement already satisfied: yacs<1.0.0,>=0.1.8 in /usr/local/lib/python3.12
[libx264 @ 0x597ac7621b00] using SAR=1/1
Requirement already satisfied: wcwidth in /usr/local/lib/python3.12/dist-packa
[libx264 @ 0x597ac7621b00] using cpu capabilities: MMX2 SSE2Fast SSSE3 SSE4.2
Requirement already satisfied: beautifulsoup4 in /usr/local/lib/python3.12/dis
[libx264 @ 0x597ac7621b00] profile High, level 3.1, 4:2:0, 8-bit
Requirement already satisfied: fsspec>=2023.5.0 in /usr/local/lib/python3.12/c
[libx264 @ 0x597ac7621b00] 264 - core 163 r3060 5db6aa6 - H.264/MPEG-4 AVC cod
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/py
Output #0, mov, to '/content/output_video_with_ASTAR_path_FOR_MAC.mov':
Requirement already satisfied: hf-xet<2.0.0,>=1.1.3 in /usr/local/lib/python3.
  Metadata:
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.12/c
    major_brand     : isom
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.12/dist-
    minor_version   : 512
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.12/
    compatible_brands: isomiso2mp41
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.12/
    encoder         : Lavf58.76.100
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.12/c
  Stream #0:0(und): Video: h264 (avc1 / 0x31637661), yuv420p(progressive), 128
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.
    Metadata:
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.12/dist-
      handler_name    : VideoHandler
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.12/dis
      vendor_id       : [0][0][0][0]
Requirement already satisfied: charset_normalizer<4,>=2 in /usr/local/lib/pyth
      encoder         : Lavc58.134.100 libx264
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.12/dist-
    Side data:
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.12
      cpb: bitrate max/min/avg: 0/0/0 buffer size: 0 vbv_delay: N/A
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.12
frame=  579 fps= 15 q=-1.0 Lsize=    8262kB time=00:00:19.86 bitrate=3407.4kb
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.12/dist
video:8255kB audio:0kB subtitle:0kB other streams:0kB global headers:0kB muxin
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.
[libx264 @ 0x597ac7621b00] frame I:3     Avg QP:21.20  size: 58440
Requirement already satisfied: setuptools in /usr/local/lib/python3.12/dist-pa
[libx264 @ 0x597ac7621b00] frame P:287   Avg QP:24.15  size: 20364
Requirement already satisfied: sympy>=1.13.3 in /usr/local/lib/python3.12/dist
[libx264 @ 0x597ac7621b00] frame B:289   Avg QP:27.17  size:  8417
Requirement already satisfied: networkx in /usr/local/lib/python3.12/dist-pack
[libx264 @ 0x597ac7621b00] consecutive B-frames: 29.0% 10.7%  7.8% 52.5%
[libx264 @ 0x597ac7621b00] mb I  I16..4: 17.7% 68.5% 13.7%
[libx264 @ 0x597ac7621b00] mb P  I16..4:  3.8% 10.1%  1.8%  P16..4: 42.6% 14.9
[libx264 @ 0x597ac7621b00] mb B  I16..4:  1.0%  2.8%  0.4%  B16..8: 45.3%  8.0
[libx264 @ 0x597ac7621b00] 8x8 transform intra:65.3% inter:66.6%
[libx264 @ 0x597ac7621b00] coded y,uvDC,uvAC intra: 41.1% 43.2% 10.1% inter: 1
[libx264 @ 0x597ac7621b00] i16 v,h,dc,p: 27% 40% 19% 15%
[libx264 @ 0x597ac7621b00] i8 v,h,dc,ddl,ddr,vr,hd,vl,hu: 17% 25% 45%  2%  2%
[libx264 @ 0x597ac7621b00] i4 v,h,dc,ddl,ddr,vr,hd,vl,hu: 21% 31% 18%  4%  5%
[libx264 @ 0x597ac7621b00] i8c dc,h,v,p: 51% 28% 18%  2%
[libx264 @ 0x597ac7621b00] Weighted P-Frames: Y:0.3% UV:0.0%
[libx264 @ 0x597ac7621b00] ref P L0: 67.4% 15.2% 12.8%  4.6%  0.0%
[libx264 @ 0x597ac7621b00] ref B L0: 89.7%  8.5%  1.8%
[libx264 @ 0x597ac7621b00] ref B L1: 96.8%  3.2%
[libx264 @ 0x597ac7621b00] kb/s:3386.75

--- Conversion Complete! ---
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.12/dist-pack
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.12/dist-pack
Your new file is ready to download: /content/output_video_with_ASTAR_path_POR
Requirement already satisfied: mpmath<1.4,>=1.1.0 in /usr/local/lib/python3.12
Requirement already satisfied: soupsieve>1.2 in /usr/local/lib/python3.12/dist
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.12/d
```

```
# --- Define Grid Scaling ---
GRID_SCALE_FACTOR = 16
GRID_WIDTH = 1280 // GRID_SCALE_FACTOR
GRID_HEIGHT = 720 // GRID_SCALE_FACTOR

# --- Define Region of Interest (ROI) ---
Y_CUTOFF = 450 # Ignores dashboard
X_LEFT_CUTOFF = 180 # Ignores side mirror
X_RIGHT_CUTOFF = 1100 # Ignores rear-view mirror

# --- Open Video ---
cap = cv2.VideoCapture(INPUT_VIDEO_PATH)

if not cap.isOpened():
    print(f"Error: Could not open video file {INPUT_VIDEO_PATH}")
else:
    if tracker is None:
        print("Warning: Tracker was not initialized. Proceeding without tracki

    frame_width, frame_height = 1280, 720
    fps = int(cap.get(cv2.CAP_PROP_FPS))

    # We will use 'mp4v' to create the file reliably
    fourcc = cv2.VideoWriter_fourcc(*'mp4v')
    OUTPUT_VIDEO_PATH = 'output_video_with_ASTAR_path.mp4'

    out = cv2.VideoWriter(OUTPUT_VIDEO_PATH, fourcc, fps, (frame_width, frame_

    print(f"--- PROCESSING: Running Modules 1 & 2 (Perception + Planning) ---"

    frame_count = 0
    while cap.isOpened():
        ret, frame = cap.read()
        if not ret: break

        # Force resize frame
        frame = cv2.resize(frame, (frame_width, frame_height))
        image_pil = Image.fromarray(cv2.cvtColor(frame, cv2.COLOR_BGR2RGB))
        annotated_frame = frame.copy()

        # === MODULE 1: PERCEPTION ===
        inputs_midas = processor_midas(images=image_pil, return_tensors="pt").
        with torch.no_grad(): outputs_midas = model_midas(**inputs_midas)
        predicted_depth = outputs_midas.predicted_depth
        prediction_midas = torch.nn.functional.interpolate(predicted_depth.uns
        depth_map_numpy = prediction_midas.cpu().numpy()

        results_yolo = model_yolo(frame, stream=True, verbose=False)

        detections = []
        for r in results_yolo:
            for box in r.boxes:
                x1, y1, x2, y2 = box.xyxy[0].cpu().numpy()
                conf = box.conf[0].cpu().item()
                cls_id = box.cls[0].cpu().item()
                # --- ROI FIX ---
                cx = (x1 + x2) / 2
                cy = (y1 + y2) / 2
                if cy < Y_CUTOFF and cx > X_LEFT_CUTOFF and cx < X_RIGHT_CUTOF
                    detections.append(
                        [int(x1), int(y1), int(x2), int(y2), conf, cls_id]
                    )

        detections_np = np.array(detections)

        tracks = np.empty((0, 8))
        if tracker is not None and len(detections_np) > 0:
            tracks = tracker.update(detections_np, frame)

        # === MODULE 2: PLANNING ===
        inputs_seg = seg_processor(images=image_pil, return_tensors="pt").to(d
        with torch.no_grad(): outputs_seg = seg_model(**inputs_seg)
        logits = outputs_seg.logits
        seg_map = torch.nn.functional.interpolate(logits, size=frame.shape[:2]
        road_mask = (seg_map == 0).astype(np.uint8)

        cost_map_grid = cv2.resize(road_mask, (GRID_WIDTH, GRID_HEIGHT), inter
        cost_map_grid = 1 - cost_map_grid # 1=obstacle, 0=drivable

        if tracker is not None:
            for track in tracks:
                x1, y1, x2, y2, track_id, conf, cls_id, _ = track
                class_name = model_yolo.names[int(cls_id)]

                if class_name in ['car', 'truck', 'bus', 'cyclist']:
                    x1_g, y1_g = int(x1/GRID_SCALE_FACTOR), int(y1/GRID_SCALE_
                    x2_g, y2_g = int(x2/GRID_SCALE_FACTOR), int(y2/GRID_SCALE_
                    if 0 <= y1_g < GRID_HEIGHT and 0 <= y2_g < GRID_HEIGHT and
                        cost_map_grid[y1_g:y2_g, x1_g:x2_g] = 1

        # --- FINAL FIX: Dynamically find a valid start node ---
        start_node = None
        # Search from 10 pixels up (to be safe) to the middle of the screen
        for y_start in range(GRID_HEIGHT - 10, GRID_HEIGHT // 2, -1):
            if cost_map_grid[y_start, GRID_WIDTH // 2] == 0: # Is this pixel d
                start_node = (GRID_WIDTH // 2, y_start) # Found it!
                break # Stop searching
        # --- END OF FIX ---

        goal_node = None
        for y in range(GRID_HEIGHT // 2, 0, -1):
            if cost_map_grid[y, GRID_WIDTH // 2] == 0: goal_node = (GRID_WIDTH
        if goal_node is None: goal_node = (GRID_WIDTH // 2, GRID_HEIGHT // 2)

        path = None
        # Only run A* if we successfully found a valid start node
        if start_node is not None:
            path = find_path_a_star(cost_map_grid, start_node, goal_node)

        # === VISUALIZATION (Module 1 + 2) ===

        road_overlay = np.zeros_like(annotated_frame, dtype=np.uint8)
        road_overlay[road_mask == 1] = (0, 255, 0)
        annotated_frame = cv2.addWeighted(annotated_frame, 1.0, road_overlay,
        if tracker is not None:
            for track in tracks:
                x1, y1, x2, y2, track_id, conf, cls_id, _ = track
                x1, y1, x2, y2, track_id = map(int, [x1, y1, x2, y2, track_id]
                class_name = model_yolo.names[int(cls_id)]

                if class_name in ['car', 'truck', 'bus', 'cyclist', 'stop sign

                    cx, cy = int((x1 + x2) / 2), int((y1 + y2) / 2)
                    depth_value = 0
                    if 0 <= cy < depth_map_numpy.shape[0] and 0 <= cx < depth_
                        dist = 1 / depth_map_numpy[cy, cx]
                        depth_value = dist if dist < 1000 and dist > 0 else 0

                    label = f"ID: {track_id} {class_name} {depth_value:.2f}m"
                    cv2.rectangle(annotated_frame, (x1, y1), (x2, y2), (255, 0
                    (w, h), _ = cv2.getTextSize(label, cv2.FONT_HERSHEY_SIMPLE
                    cv2.rectangle(annotated_frame, (x1, y1 - h - 10), (x1 + w,
                    cv2.putText(annotated_frame, label, (x1, y1 - 5), cv2.FONT

        # Draw A* Path (Red Line)
        if path:
            path_scaled = (np.array(path) * GRID_SCALE_FACTOR + GRID_SCALE_FAC
            cv2.polylines(annotated_frame, [path_scaled], isClosed=False, colo

        out.write(annotated_frame)

        frame_count += 1
        if frame_count % 50 == 0: print(f"Processed {frame_count} frames...")

    # Release everything
    cap.release()
    out.release()
    cv2.destroyAllWindows()

    print(f"--- Module 1+2 Processing Complete! ---")
    print(f"Output video saved as: {OUTPUT_VIDEO_PATH}")
```

```
--- PROCESSING: Running Modules 1 & 2 (Perception + Planning) ---
Processed 50 frames...
Processed 100 frames...
Processed 150 frames...
Processed 200 frames...
Processed 250 frames...
Processed 300 frames...
Processed 350 frames...
Processed 400 frames...
Processed 450 frames...
Processed 500 frames...
Processed 550 frames...
--- Module 1+2 Processing Complete! ---
Output video saved as: output_video_with_ASTAR_path.mp4
```

```
from google.colab import files
import os

INPUT_FILE = 'output_video_with_ASTAR_path.mp4'
OUTPUT_FILE_FOR_MAC = 'output_video_with_ASTAR_path_FOR_MAC.mov'

# Check if the input file exists
if not os.path.exists(INPUT_FILE):
    print(f"--- ERROR ---")
```

```
        print(f"--- ERROR ---")
        print(f"The input file '{INPUT_FILE}' was not found.")
    else:
        print(f"Converting '{INPUT_FILE}' to a Mac-compatible .mov file...")

        # Use FFmpeg to re-encode the video with the H.264 (avc1) codec
        !ffmpeg -i "{INPUT_FILE}" -c:v libx264 -pix_fmt yuv420p -f mov "{OUTPUT_FIL

        print(f"\n--- Conversion Complete! ---")
        print(f"Your new file is ready to download: {OUTPUT_FILE_FOR_MAC}")

        # Download the new file
        files.download(OUTPUT_FILE_FOR_MAC)
```

Module three control

```
# Cell 1: Installs
!pip install ultralytics transformers timm boxmot
```

```
Requirement already satisfied: huggingface-hub<1.0,>=0.34.0 in /usr/local/lib/
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.12/di
Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib/python3.12/
Requirement already satisfied: tokenizers<=0.23.0,>=0.22.0 in /usr/local/lib/p
Requirement already satisfied: safetensors>=0.4.3 in /usr/local/lib/python3.12
Requirement already satisfied: tqdm>=4.27 in /usr/local/lib/python3.12/dist-pa
Requirement already satisfied: click>=8.1.8 in /usr/local/lib/python3.12/dist-
Requirement already satisfied: filterpy<2.0.0,>=1.4.5 in /usr/local/lib/python
Requirement already satisfied: ftfy<7.0.0,>=6.1.3 in /usr/local/lib/python3.12
Requirement already satisfied: gdown<6.0.0,>=5.1.0 in /usr/local/lib/python3.1
Requirement already satisfied: lapx<1.0.0,>=0.5.5 in /usr/local/lib/python3.12
Requirement already satisfied: loguru<1.0.0,>=0.7.2 in /usr/local/lib/python3.
Requirement already satisfied: pandas<3.0.0,>=2.0.0 in /usr/local/lib/python3.
Requirement already satisfied: scikit-learn<2.0.0,>=1.3.0 in /usr/local/lib/py
Requirement already satisfied: yacs<1.0.0,>=0.1.8 in /usr/local/lib/python3.12
Requirement already satisfied: wcwidth in /usr/local/lib/python3.12/dist-packa
Requirement already satisfied: beautifulsoup4 in /usr/local/lib/python3.12/dis
Requirement already satisfied: fsspec>=2023.5.0 in /usr/local/lib/python3.12/d
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/py
Requirement already satisfied: hf-xet<2.0.0,>=1.1.3 in /usr/local/lib/python3.
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.12/d
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.12/dist-
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.12/
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.12/
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.12/d
Requirement already satisfied: nvidia-cufft-cu12==11.3.0.4 in /usr/local/lib/p
Converting 'output_video_with_ASTAR_path.mp4' to a Mac-compatible .mov file...
Requirement already satisfied: nvidia-curand-cu12==10.3.7.77 in /usr/local/lib
ffmpeg version 4.4.2-0ubuntu0.22.04.1 Copyright (c) 2000-2021 the FFmpeg deve
Requirement already satisfied: nvidia-cusolver-cu12==11.7.1.2 in /usr/local/li
  built with gcc 11 (Ubuntu 11.2.0-19ubuntu1)
Requirement already satisfied: nvidia-cusparse-cu12==12.5.4.2 in /usr/local/li
  configuration: --prefix=/usr --extra-version=0ubuntu0.22.04.1 --toolchain=ha
Requirement already satisfied: nvidia-cusparselt-cu12==0.7.1 in /usr/local/lib
  libavutil      56. 70.100 / 56. 70.100
Requirement already satisfied: nvidia-nccl-cu12==2.27.3 in /usr/local/lib/pyth
  libavcodec     58.134.100 / 58.134.100
Requirement already satisfied: nvidia-nvtx-cu12==12.6.77 in /usr/local/lib/pyt
  libavformat    58. 76.100 / 58. 76.100
Requirement already satisfied: nvidia-nvjitlink-cu12==12.6.85 in /usr/local/li
  libavdevice    58. 13.100 / 58. 13.100
Requirement already satisfied: nvidia-cufile-cu12==1.11.1.6 in /usr/local/lib/
  libavfilter     7.110.100 /  7.110.100
Requirement already satisfied: triton==3.4.0 in /usr/local/lib/python3.12/dist
  libswscale      5.  9.100 /  5.  9.100
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.12/dist-pack
  libswresample   3.  9.100 /  3.  9.100
Requirement already satisfied: mpmath<1.4,>=1.1.0 in /usr/local/lib/python3.12
  libpostproc    55.  9.100 / 55.  9.100
Requirement already satisfied: soupsieve>1.2 in /usr/local/lib/python3.12/dist
Input #0, mov,mp4,m4a,3gp,3g2,mj2, from 'output_video_with_ASTAR_path.mp4':
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.12/di
  Metadata:
Requirement already satisfied: PySocks!=1.5.7,>=1.5.6 in /usr/local/lib/python
    major_brand     : isom
    minor_version   : 512
```

```
# --- Imports & Model Loading (Cell 2 - FINAL FIX 2) ---

import cv2
from ultralytics import YOLO
from google.colab import files
import torch
from transformers import DPTForDepthEstimation, DPTImageProcessor, SegformerFo
import numpy as np
from PIL import Image
from boxmot import create_tracker # Using your correct import
import os
import heapq
import time

# --- Setup Device (use GPU if available) ---
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(f"Using device: {device}")

# --- 1. Load YOLOv8 Model ---
model_yolo = YOLO('yolov8n.pt')

# --- 2. Load MiDaS Depth Estimation Model ---
print("Loading MiDaS model...")
# The "not initialized" warning is normal and can be ignored.
processor_midas = DPTImageProcessor.from_pretrained("Intel/dpt-large")
model_midas = DPTForDepthEstimation.from_pretrained("Intel/dpt-large")
model_midas.to(device)
print("MiDaS model loaded.")

# --- 3. Initialize the Tracker (Using ByteTrack) ---
tracker = None # Initialize tracker to None
try:
    print("Initializing ByteTrack tracker...")
    # --- THIS IS THE FIX ---
    # We are switching from 'deepsort' (which is broken in the library)
    # to 'bytetrack', which is included and works.
    tracker = create_tracker(
        tracker_type='bytetrack', # <-- THE FIX
        device=device,
        half=True
    )
    if tracker is not None:
        print("--- Tracker initialized SUCCESSFULLY. ---")
    else:
        print("--- Tracker initialization FAILED (returned None). ---")
```

```
except Exception as e:
    # This will catch any errors
    print(f"--- AN ERROR OCCURRED during tracker initialization: {e} ---")

# --- 4. Load SegFormer Segmentation Model ---
print("Loading SegFormer model for Drivable Area...")
seg_processor = SegformerImageProcessor.from_pretrained("nvidia/segformer-b0-f
seg_model = SegformerForSemanticSegmentation.from_pretrained("nvidia/segformer
seg_model.to(device)
print("SegFormer model loaded.")



# --- Define Video Paths ---
INPUT_VIDEO_PATH = 'My Movie.mp4'
OUTPUT_VIDEO_PATH = 'output_video_FINAL_with_control.mp4'
```

```
Using device: cuda
Loading MiDaS model...
Some weights of DPTForDepthEstimation were not initialized from the model check
You should probably TRAIN this model on a down-stream task to be able to use it
2025-11-16 07:54:26.735 | MainProcess/MainThread | INFO     | /usr/local/lib/py
2025-11-16 07:54:26.736 | MainProcess/MainThread | INFO     | /usr/local/lib/py
2025-11-16 07:54:26.744 | MainProcess/MainThread | INFO     | /usr/local/lib/py
2025-11-16 07:54:26.744 | MainProcess/MainThread | INFO     | /usr/local/lib/py
2025-11-16 07:54:26.744 | MainProcess/MainThread | INFO     | /usr/local/lib/py
2025-11-16 07:54:26.744 | MainProcess/MainThread | INFO     | /usr/local/lib/py
2025-11-16 07:54:26.745 | MainProcess/MainThread | INFO     | /usr/local/lib/py
2025-11-16 07:54:26.745 | MainProcess/MainThread | INFO     | /usr/local/lib/py
2025-11-16 07:54:26.745 | MainProcess/MainThread | INFO     | /usr/local/lib/py
2025-11-16 07:54:26.745 | MainProcess/MainThread | INFO     | /usr/local/lib/py
2025-11-16 07:54:26.745 | MainProcess/MainThread | SUCCESS  | /usr/local/lib/py
MiDaS model loaded.
Initializing ByteTrack tracker...
--- Tracker initialized SUCCESSFULLY. ---
Loading SegFormer model for Drivable Area...
/usr/local/lib/python3.12/dist-packages/transformers/image_processing_base.py:4
  image_processor = cls(**image_processor_dict)
SegFormer model loaded.
```

```
# --- A* Pathfinding Algorithm ---
class Node:
    def __init__(self, x, y, g=0, h=0, parent=None):
        self.x = x
        self.y = y
        self.g = g; self.h = h; self.f = g + h
        self.parent = parent
    def __lt__(self, other):
        return self.f < other.f

def find_path_a_star(cost_map, start, goal):
    neighbors = [(0, 1), (0, -1), (1, 0), (-1, 0), (1, 1), (1, -1), (-1, 1), (
    close_list = set()
    open_list = []
    start_node = Node(start[0], start[1], 0, heuristic(start, goal))
    goal_node = Node(goal[0], goal[1])
    heapq.heappush(open_list, start_node)

    while open_list:
        current_node = heapq.heappop(open_list)
        if (current_node.x, current_node.y) == (goal_node.x, goal_node.y):
            path = [];
            while current_node: path.append((current_node.x, current_node.y));
            return path[::-1]
        close_list.add((current_node.x, current_node.y))
        for dx, dy in neighbors:
            neighbor_x, neighbor_y = current_node.x + dx, current_node.y + dy
            if not (0 <= neighbor_x < cost_map.shape[1] and 0 <= neighbor_y <
            if cost_map[neighbor_y, neighbor_x] == 1: continue
            if (neighbor_x, neighbor_y) in close_list: continue
            g_cost = current_node.g + (1.414 if dx != 0 and dy != 0 else 1)
            h_cost = heuristic((neighbor_x, neighbor_y), goal)
            neighbor_node = Node(neighbor_x, neighbor_y, g_cost, h_cost, curre
            if any(n.x == neighbor_x and n.y == neighbor_y and n.f <= neighbor
            heapq.heappush(open_list, neighbor_node)
    return None

def heuristic(a, b):
    return abs(a[0] - b[0]) + abs(a[1] - b[1])

print("A* Pathfinding functions defined.")
```

```
A* Pathfinding functions defined.
```

```
# --- A* Pathfinding Algorithm Helper Functions ---
# (Assumes Node class and find_path_a_star are already defined in a previous ce

# --- MODULE 3: CONTROL (Helper Functions) ---

# 1. PID Controller Class
class PIDController:
    def __init__(self, Kp, Ki, Kd, set_point, output_limits):
        self.Kp = Kp; self.Ki = Ki; self.Kd = Kd
        self.set_point = set_point
        self.output_limits = output_limits
        self._prev_error = 0; self._integral = 0

    def update(self, measurement, dt):
        if dt == 0: return self.output_limits[0]
        error = self.set_point - measurement
        P_out = self.Kp * error
        self._integral += error * dt
        self._integral = np.clip(self._integral, self.output_limits[0], self.ou
        I_out = self.Ki * self._integral
        derivative = (error - self._prev_error) / dt
        D_out = self.Kd * derivative
        output = np.clip(P_out + I_out + D_out, self.output_limits[0], self.out
        self._prev_error = error
        return output

# 2. Behavioral Logic Function (WITH DEPTH FIX)
def behavioral_planner(tracks, depth_map_numpy, cost_map_grid, grid_width, grid_
    target_speed_kph = 40.0 # Default speed
    state = "DRIVE"
    lane_center_x = grid_width // 2
    lane_width_pixels = grid_width * 0.20
    lane_x1 = int(lane_center_x - (lane_width_pixels / 2))
    lane_x2 = int(lane_center_x + (lane_width_pixels / 2))

    # --- DEPTH LOGIC FIX ---
    # MiDaS raw output: LARGER value = CLOSER object
    closest_obstacle_dist = 0.0 # 0 is "infinitely far"
    closest_obstacle_dist = 0.0 # 0 is "infinitely far"
    obstacle_detected = False

    for track in tracks:
        if len(track) >= 7:
            x1, y1, x2, y2, _, _, cls_id, *_ = track
            class_name = model_yolo.names[int(cls_id)]

            if class_name in ['car', 'truck', 'bus', 'cyclist']:
                x1_grid = int(x1 / grid_scale_factor)
                x2_grid = int(x2 / grid_scale_factor)

                if max(x1_grid, lane_x1) < min(x2_grid, lane_x2):
                    obstacle_detected = True
                    cx, cy = int((x1 + x2) / 2), int((y1 + y2) / 2)
                    if 0 <= cy < depth_map_numpy.shape[0] and 0 <= cx < depth_ma
                        # Get the RAW inverse depth. Larger = Closer
                        dist = depth_map_numpy[cy, cx]
                        if dist > closest_obstacle_dist: # Find the largest valu
                            closest_obstacle_dist = dist

    if obstacle_detected:
        # We must tune these values. A value of 4-6 is "close"
        # These values are tuned for the video
        if closest_obstacle_dist > 5.5: # Very close (large inverse depth value
            state = "STOP"
            target_speed_kph = 0.0
        elif closest_obstacle_dist > 4.0: # Getting closer
            state = "FOLLOW"
            target_speed_kph = 20.0
    # --- END OF FIX ---

    return target_speed_kph, state

# --- Grid Scaling ---
GRID_SCALE_FACTOR = 16
GRID_WIDTH = 1280 // GRID_SCALE_FACTOR
GRID_HEIGHT = 720 // GRID_SCALE_FACTOR

# --- ROI Definition ---
Y_CUTOFF = 450 # Ignores dashboard
X_LEFT_CUTOFF = 180 # Ignores side mirror
X_RIGHT_CUTOFF = 1100 # Ignores rear-view mirror

# --- Initialize Control Systems ---
steering_pid = PIDController(Kp=0.8, Ki=0.1, Kd=0.2, set_point=0, output_limits=
speed_pid = PIDController(Kp=0.1, Ki=0.01, Kd=0.05, set_point=0, output_limits=

# --- Simulation Variables ---
current_speed_kph = 0.0
last_frame_time = time.time()

# --- Open Video ---
cap = cv2.VideoCapture(INPUT_VIDEO_PATH)

if not cap.isOpened():
    print(f"Error: Could not open video file {INPUT_VIDEO_PATH}")
else:
    if tracker is None:
        print("Warning: Tracker was not initialized. Proceeding without trackin
    frame_width, frame_height = 1280, 720
    fps = int(cap.get(cv2.CAP_PROP_FPS))

    # Create the 'mp4v' file. We will convert it after.
    fourcc = cv2.VideoWriter_fourcc(*'mp4v')
    OUTPUT_VIDEO_PATH = 'output_video_FINAL_with_control.mp4'

    out = cv2.VideoWriter(OUTPUT_VIDEO_PATH, fourcc, fps, (frame_width, frame_h

    print(f"--- FINAL PROCESSING: ALL MODULES ENABLED ---")

    frame_count = 0
    while cap.isOpened():
        ret, frame = cap.read()
        if not ret: break

        current_frame_time = time.time()
        dt = current_frame_time - last_frame_time
        if dt == 0: dt = 1/fps
        last_frame_time = current_frame_time

        frame = cv2.resize(frame, (frame_width, frame_height))
        image_pil = Image.fromarray(cv2.cvtColor(frame, cv2.COLOR_BGR2RGB))
        annotated_frame = frame.copy()

        # === MODULE 1: PERCEPTION ===
        inputs_midas = processor_midas(images=image_pil, return_tensors="pt").t
        with torch.no_grad(): outputs_midas = model_midas(**inputs_midas)
        predicted_depth = outputs_midas.predicted_depth
        prediction_midas = torch.nn.functional.interpolate(predicted_depth.unsq
        depth_map_numpy = prediction_midas.cpu().numpy()

        results_yolo = model_yolo(frame, stream=True, verbose=False)
        detections = []
        for r in results_yolo:
            for box in r.boxes:
                x1, y1, x2, y2 = box.xyxy[0].cpu().numpy()
                conf = box.conf[0].cpu().item()
                cls_id = box.cls[0].cpu().item()
                cx = (x1 + x2) / 2
                cy = (y1 + y2) / 2
                # Filter by ROI
                if cy < Y_CUTOFF and cx > X_LEFT_CUTOFF and cx < X_RIGHT_CUTOFF
                    detections.append([int(x1), int(y1), int(x2), int(y2), conf
        detections_np = np.array(detections)

        tracks = np.empty((0, 8))
        if tracker is not None and len(detections_np) > 0:
            tracks = tracker.update(detections_np, frame)

        # === MODULE 2: PLANNING ===
        inputs_seg = seg_processor(images=image_pil, return_tensors="pt").to(dev
        with torch.no_grad(): outputs_seg = seg_model(**inputs_seg)
        logits = outputs_seg.logits
        seg_map = torch.nn.functional.interpolate(logits, size=frame.shape[:2],
        road_mask = (seg_map == 0).astype(np.uint8)
        cost_map_grid = cv2.resize(road_mask, (GRID_WIDTH, GRID_HEIGHT), interp
        cost_map_grid = 1 - cost_map_grid

        if tracker is not None:
            for track in tracks:
```

```
                    if len(track) >= 7:
                        x1, y1, x2, y2, _, _, cls_id, *_ = track
                        if model_yolo.names[int(cls_id)] in ['car', 'truck', 'bus',
                            x1_g, y1_g = int(x1/GRID_SCALE_FACTOR), int(y1/GRID_SCA
                            x2_g, y2_g = int(x2/GRID_SCALE_FACTOR), int(y2/GRID_SCA
                            x1_g, y1_g = max(0, x1_g), max(0, y1_g)
                            x2_g, y2_g = min(GRID_WIDTH-1, x2_g), min(GRID_HEIGHT-1
                            cost_map_grid[y1_g:y2_g, x1_g:x2_g] = 1

            # --- STEERING / A* FIX: Dynamically find a valid start node ---
            start_node = None
            for y_start in range(GRID_HEIGHT - 10, GRID_HEIGHT // 2, -1): # Search
                if cost_map_grid[y_start, GRID_WIDTH // 2] == 0:
                    start_node = (GRID_WIDTH // 2, y_start)
                    break
            # --- END OF FIX ---

            goal_node = None
            for y in range(GRID_HEIGHT // 2, 0, -1):
                if cost_map_grid[y, GRID_WIDTH // 2] == 0: goal_node = (GRID_WIDTH
            if goal_node is None: goal_node = (GRID_WIDTH // 2, GRID_HEIGHT // 2)

            path = None
            if start_node is not None: # Only run if start_node is valid
                path = find_path_a_star(cost_map_grid, start_node, goal_node)

            # === MODULE 3: CONTROL ===
            target_speed_kph, state = behavioral_planner(tracks, depth_map_numpy, c
            speed_pid.set_point = target_speed_kph
            speed_control_output = speed_pid.update(current_speed_kph, dt)
            throttle = max(0, speed_control_output)
            brake = max(0, -speed_control_output)

            steering_angle = 0.0
            if path and len(path) > 1:
                target_point_index = min(5, len(path) - 1)
                target_x, target_y = path[target_point_index]
                steering_error = target_x - start_node[0]
                steering_pid.set_point = 0
                steering_angle = steering_pid.update(steering_error, dt)

            current_speed_kph += (throttle * 10) * dt
            current_speed_kph -= (brake * 30) * dt
            current_speed_kph = max(0, current_speed_kph)

            # --- FINAL VISUALIZATION ---
            road_overlay = np.zeros_like(annotated_frame, dtype=np.uint8)
            road_overlay[road_mask == 1] = (0, 255, 0)
            annotated_frame = cv2.addWeighted(annotated_frame, 1.0, road_overlay, 0

            if tracker is not None:
                for track in tracks:
                    if len(track) >= 7:
                        x1, y1, x2, y2, track_id, _, cls_id, *_ = track
                        x1, y1, x2, y2, track_id = map(int, [x1, y1, x2, y2, track_

                        if model_yolo.names[int(cls_id)] in ['car', 'truck', 'bus',
                            cv2.rectangle(annotated_frame, (x1, y1), (x2, y2), (255
                            label = f"ID: {track_id} {model_yolo.names[int(cls_id)]
                            (w, h), _ = cv2.getTextSize(label, cv2.FONT_HERSHEY_SIM
                            cv2.rectangle(annotated_frame, (x1, y1 - h - 10), (x1 +
                            cv2.putText(annotated_frame, label, (x1, y1 - 5), cv2.F

            if path:
                path_scaled = (np.array(path) * GRID_SCALE_FACTOR + GRID_SCALE_FACT
                cv2.polylines(annotated_frame, [path_scaled], isClosed=False, color

            cv2.rectangle(annotated_frame, (10, 10), (350, 180), (0,0,0), -1)
            cv2.putText(annotated_frame, f"STATE: {state}", (20, 40), cv2.FONT_HERS
            cv2.putText(annotated_frame, f"SPEED: {current_speed_kph:.1f} kph", (20
            cv2.putText(annotated_frame, f"TARGET: {target_speed_kph:.1f} kph", (20
            cv2.putText(annotated_frame, f"STEERING: {steering_angle:.1f} deg", (20
            throttle_bar = int(throttle * 100)
            brake_bar = int(brake * 100)
            cv2.putText(annotated_frame, f"THROTTLE: {throttle_bar}%", (200, 105),
            cv2.putText(annotated_frame, f"BRAKE: {brake_bar}%", (200, 140), cv2.FO

            out.write(annotated_frame)

            frame_count += 1
            if frame_count % 50 == 0: print(f"Processed {frame_count} frames...")

        # Release everything
        cap.release()
        out.release()
        cv2.destroyAllWindows()

        print(f"--- PROJECT COMPLETE! ---")
        print(f"Final video saved as: {OUTPUT_VIDEO_PATH}")
```

```
--- FINAL PROCESSING: ALL MODULES ENABLED ---
Processed 50 frames...
Processed 100 frames...
Processed 150 frames...
Processed 200 frames...
Processed 250 frames...
Processed 300 frames...
Processed 350 frames...
Processed 400 frames...
Processed 450 frames...
Processed 500 frames...
Processed 550 frames...
--- PROJECT COMPLETE! ---
Final video saved as: output_video_FINAL_with_control.mp4
```

```
from google.colab import files
import os

INPUT_FILE = '/content/output_video_FINAL_with_control.mp4'
OUTPUT_FILE_FOR_MAC = '/content/output_video_FOR_MAC.mov' # This is the file yo

# First, check if the input file was created correctly
if not os.path.exists(INPUT_FILE):
    print(f"--- ERROR ---")
    print(f"The input file '{INPUT_FILE}' was not found.")
    print("This means your main processing loop (Cell 5) failed to run or crash
else:
    print(f"Converting '{INPUT_FILE}' to a Mac-compatible .mov file...")

    # This command uses FFmpeg to re-encode the video with:
    # -c:v libx264: The H.264 (avc1) codec QuickTime needs
    # -pix_fmt yuv420p: The pixel format QuickTime needs
```

```
        # -f mov: The .mov file container Apple prefers

        !ffmpeg -i "{INPUT_FILE}" -c:v libx264 -pix_fmt yuv420p -f mov "{OUTPUT_FIL

        print(f"\n--- Conversion Complete! ---")
        print(f"Your new file is ready to download: {OUTPUT_FILE_FOR_MAC}")

        # Download the new file
        files.download(OUTPUT_FILE_FOR_MAC)
```